Curso 2012/13
CIENCIAS Y TECNOLOGÍAS/23
I.S.B.N.: 978-84-15910-90-9

SANTIAGO BASALDÚA LEMARCHAND

Monte-Carlo tree search using expert knowledge: an application to computer go and human genetics

Directores
J. MARCOS MORENO VEGA
CARLOS A. FLORES INFANTE

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

Universidad de La Laguna

Abstract

Monte-Carlo Tree Search Using Expert Knowledge: An Application to Computer Go and Human Genetics

During the years in which the research described in this PhD dissertation was done, Monte-Carlo Tree Search has become the preeminent algorithm in many AI and computer science fields. This dissertation analyzes how expert knowledge and also online learned knowledge can be used to enhance the search. The work describes two different implementations: as a two player search in computer *go* and as an optimization method in human genetics. It is established that in large problems MCTS has to be combined with domain specific or online learned knowledge to improve its strength. This work analyzes different successful ideas about how to do it, the resulting findings and their implications, hence improving our insight of MCTS. The main contributions to the field are: an analytical mathematical model improving the understanding of simulations, a problem definition and a framework including code and data to compare algorithms in human genetics and three successful implementations: in the field of 19x19 *go* openings named M-eval, in the field of learning playouts and in the field of genetic etiology. Also, an open source integer representation of proportions as Win/Loss States (WLS), a negative result in the field of playouts, an unexpected finding of a possible problem in optimization and further insight on the limitations of MCTS are worth mentioning. With the exception of some background materials introducing the game of *go* and MCTS, the dissertation is entirely written in English.

Incorporación de conocimiento experto a la búsqueda en árbol mediante procesos estocásticos y su aplicación en el juego abstracto del *go* y en genética humana

Durante el periodo de desarrollo de esta tesis Monte-Carlo Tree Search (MCTS la búsqueda en árbol mediante procesos estocásticos) se ha convertido en el algoritmo principal en muchos problemas de inteligencia artificial e informática. Esta tesis analiza la incorporación de conocimiento experto para mejorar la búsqueda. El trabajo describe dos aplicaciones: una en el juego del *go* por ordenador y otra en el campo de la genética humana. Es un hecho establecido que, en problemas complejos, MCTS requiere del apoyo de conocimiento específico o aprendido *online* para mejorar su rendimiento. Lo que este trabajo analiza son diferentes ideas de cómo hacerlo, sus resultados e implicaciones, mejorando así nuestra comprensión de MCTS. Las principales contribuciones al área son: un modelo analítico de las simulaciones que mejora la comprensión del papel de las simulaciones, un marco competitivo incluyendo código y datos para comparar métodos en etiología genética y tres aplicaciones con éxito: una en el campo de las aperturas en *go* de 19x19 llamada M-eval, otra sobre simulaciones que aprenden y una en etiología genética. Además, merece la pena destacar: un modelo para representar proporciones mediante estados llamado WLS con software libre, un resultado negativo sobre una idea para las simulaciones, el descubrimiento inesperado de un posible problema utilizando MCTS en optimización y un análisis original de las limitaciones. Con la excepción de unos materiales introductorios sobre el juego del *go* y Monte-Carlo Tree Search, la tesis está completamente escrita en inglés.

by Santiago Basaldúa Lemarchand

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The author wishes to thank:

- Milagros Gómez Martín for her constant support during the years of this research project.
- My two directors: J. Marcos Moreno Vega, Ph.D. and Carlos Flores Infante, Ph.D. for their expertise and valuable contributions to my research and the manuscript.
- The contributors to the Computer Go Mailing list which have been a valuable source of free information and discussion for my research and especially Peter Drake, Ph.D. who helped me with some papers.
- The reviewers for the published papers who have contributed to the improvement of the published papers and this manuscript.

Chapter 1. Introduction

1.1 Thesis overview for English readers

The purpose of this PhD dissertation is to study and experiment about expert knowledge, both offline learned and online learned, applied in two different implementations of Monte-Carlo Tree Search (MCTS) in the fields of computer *go* (a minimax implementation of MCTS) and human genetics (MCTS applied in optimization). All non-introductory materials are written in English, both the state of the art research in the fields and our original research. In the case of our application to human genetics, since this dissertation is intended for an audience of computer scientists, all necessary explanations about biology and genetics are included in English in 5.1.

Chapter 3 is dedicated to the description of successful methods for providing knowledge (in the form of a priori values) to the tree search including the most important ideas introduced by other authors and our original research: M-Eval. [1, 2]. Chapter 4 is dedicated to the stochastic evaluation function, known as *playout* in the field of computer *go*. It includes the most successful and widely accepted methods by other authors and our own research: WLS [3] and *learning playouts* [4]. Chapter 5 is a self-contained dissertation on the application of MCTS in the field of human genetics for an audience of computer scientists/mathematicians covering from the background to the experimental results and can also be read independently without any *go* background. All the experiments in genetics are performed using the GEM platform (described in 6.2), an original work of the main author, first implemented to research bias in case-control association studies [5] and finally, including the results of this thesis [6]. Chapter 6 describes all the software used in this thesis, which was almost completely

written by the main author. Chapter 7 contains the discussion including recent research in MCTS and chapter 8 enumerates the conclusions.

An English reader with understanding of *go*, computer *go* and Monte-Carlo Tree Search can safely jump to page 30. Otherwise, the following references may be useful:

- **About *Go***: Introduction by the American *Go* Association [7], introduction by the British *Go* Association [8], an interactive online tutorial [9], an introductory book written by the top professional player Cho Chikun [10], Wikipedia article on the game [11], a wiki site exclusively dedicated to *go* [12] and official rules in English: Chinese-style [13] and Japanese-style [14].
- About the **meaning of *Go* terms** used in this dissertation, such as *semeai*, *joseki*, *sente*, etc., Sensei's Library [12] is a very good place to find a description of those terms.
- **About Computer *Go***: Two excellent analyses on pre-MCTS computer *go* by Erik van der Werf [15] and Martin Müller [16], Wikipedia article on computer *go* [17], an updated list of computer *go* bibliography [18] and the Computer *Go* Mailing List where most researchers share news and ideas [19].
- **About Monte-Carlo Tree Search**: At first (around 2007). MCTS was still named UCT (Upper Confidence-bound for Trees). Rewarding credit to the seminal papers, UCT was first described by Levente Kocsis and Csaba Szepesvari in 2006 [20] and implemented in computer *go* by Sylvain Gelly and David Silver [21] introducing pattern-based playouts in collaboration with Yizao Wang [22]. Most important early improvements include RAVE [21, 23], progressive widening by Rémi Coulom [24] and progressive bias by Guillaume Chaslot [25]. A first description of an UCT engine including pseudo-code by Magnus Persson (author of the program Valkyria) can also be found online [26]. A modern web site dedicated to MCTS is also worth mentioning [27].

1.2 Descripción de la tesis en español

Esta tesis trata sobre la incorporación de conocimiento experto, obtenido tanto *offline* como *online*, a la búsqueda en árbol con evaluación estocástica, MCTS *(Monte-Carlo Tree Search)*. En la tesis se describen dos aplicaciones, una al juego del *go* (una aplicación minimax) y otra en el campo de la genética humana (una aplicación de optimización). Los contenidos fundamentales están descritos en inglés, tanto la descripción del

estado de la técnica en las respectivas áreas, como la descripción del trabajo de investigación realizado y las conclusiones.

En cuanto a la descripción de los fundamentos, el capítulo 2 contiene una descripción en español que abarca el juego del *go*, desgraciadamente insuficientemente conocido en España, desde su historia, las reglas y la relevancia mundial del juego. El capítulo también describe someramente algunos de los avances en *go* por ordenador hasta la aparición de MCTS. Finalmente, el capítulo describe una aplicación básica de MCTS a un problema de optimización a modo de introducción del algoritmo fundamental de la tesis, MCTS. Además, incluye un resumen en español de la implementación de MCTS en genética descrita en inglés en el capítulo 5.

Los fundamentos de biología y genética necesarios para entender la aplicación en genética humana están, únicamente en inglés, en el apartado 5.1.

El capítulo 3 está dedicado a diversos métodos de incorporación de conocimiento al árbol MCTS en forma de evaluación a priori. Incluye métodos descritos por otros autores y nuestro trabajo original M-Eval [1, 2]. El capítulo 4 está dedicado a la función de evaluación estocástica conocida como *playout* en el campo del *go* por ordenador. Incluye métodos descritos por otros autores junto con nuestra investigación original que incluye WLS [3] y *playouts* que aprenden [4]. El capítulo 5 es una descripción completa de la aplicación en el campo de la genética humana que incluye todos los fundamentos de biología y genética necesarios para ser entendida por matemáticos o informáticos. Todos los experimentos de genética se realizaron con la plataforma GEM, una obra original del autor principal, utilizada primero para investigar el sesgo en estudios de asociación casos-controles [5] y finalmente, incluyendo los resultados de la tesis [6]. El capítulo 6 describe todo el software utilizado en esta tesis y desarrollado casi totalmente por el autor. El capítulo 7 contiene la discusión de la tesis junto con una descripción de trabajos de investigación en MCTS. Finalmente, el capítulo 8, que también está el español, enumera las conclusiones de la tesis.

Chapter 2. Background (in Spanish)

2.1 Introducción al juego del *go*

2.1.1 Historia del juego del *go*

China antigua: La tradición sitúa el origen del *go*, cuyo nombre chino es *weiqi*, hace más de 4000 años en tiempos del emperador *Yao* (2357-2255 AC) o su sucesor, *Shun* (2255-2205 AC). Confucio menciona el juego en "Los Analectas" en el siglo 6 AC. Posteriormente, hay numerosos grabados y documentos que prueban su existencia. El primer tratado conocido sobre *go* tiene aproximadamente 2000 años, es el *Yi Zhi* (Esencia del *go*) escrito por *Ban Gu* (32-92 DC).

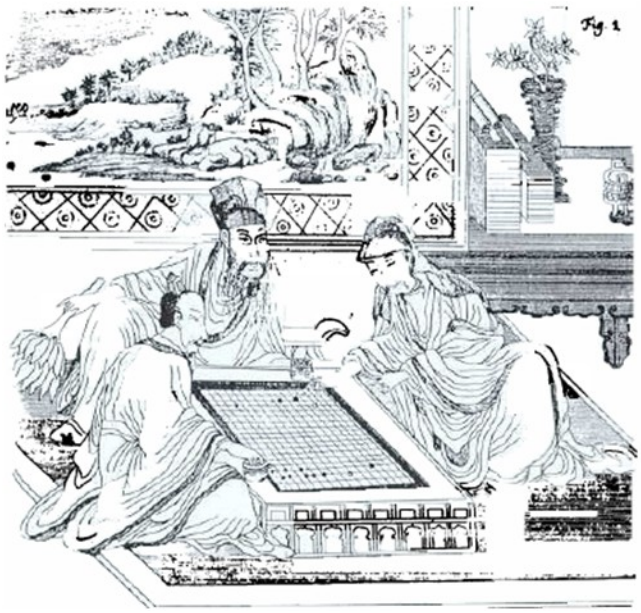

Figura 2.1. El juego del *go* en la antigua China.

Japón en el periodo Edo (1603-1868): Aunque este juego chino ya era popular en Japón desde el siglo VIII, el primer "gran maestro" de *go* es *Honinbō Sansa* (nacido *Kanō Yosaburo*) un monje budista. En 1612 funda la escuela *Honinbo* que junto con las escuelas *Yasui*, *Inoue* y *Hayashi* constituyen las cuatro grandes escuelas de este periodo. El estudio del *go* se convierte en una profesión respetada. Las escuelas conservan archivos de las partidas jugadas y en ellas se dedican únicamente al estudio del juego como un camino dentro del budismo. Además, pasa a formar parte de la educación masculina, pues éste es un periodo de poder ejercido por un comandante militar, el Shogún, y el *go* es la base para la enseñanza sobre estrategia.

Europa siglo XVIII: Leibniz publica una descripción del juego que denomina "juego chino". El artículo original en latín aparece en primer lugar en la revista berlinesa *"Miscellanea Berolinensia Ad Increme Scientiarum"* en 1710. El artículo describe las reglas del juego con una ilustración china de una partida de *go*.

III.

Godefridi Guilielmi Leibnitii

Annotatio de quibasdam Ludis; inprimis de Ludo quodam Sinico, differentiaque Scachici & Latrunculorum, & novo genere Ludi Navalis.

LE PALAMÈDE

Revue Mensuelle

DES ÉCHECS ET AUTRES JEUX.

DEUXIÈME SÉRIE. — TOME SEPTIÈME.

Paris.

1847.

NOTE

GODEFROI GUILLAUME LEIBNITZ,

Figura 2.2. Fragmento del artículo de Leibniz 1710 y su traducción al francés en 1847

Este artículo es traducido al francés en 1847 por *Alliey* y publicado en la revista «Le Palamède des échecs et autres jeux».

El *go* profesional en la actualidad (2009): Existe fundamente en Japón, Corea, China y Taiwán, pero también en EEUU. La lista de torneos con premios para el ganador superiores a 100 000 dólares norteamericanos: *Ing Cup* 500 K$, *Kisei* 365 K$, *Meijin* 330 K$, *BC Card Cup* 300 K$, *World Oza* 285 K$, *Honinbo* 280 K$, *LG Cup* 250 K$, *Samsung Cup* 200 K$, *Chunlan Cup* 170 K$, *Fujitsu Cup* 141 K$, *Judan*

126 K$, *Tengen* 122 K$ y *Oza* 118 K$ es suficientemente amplia para que el *go* sea una profesión a la que se aspira desde la infancia. Algunos jugadores, como el coreano Lee Chang-ho, nacido en 1975 y que ya ha ganado 132 torneos profesionales, son ídolos en sus países y para los aficionados al *go* del mundo entero (datos de 2009).

2.1.2 Las reglas básicas del *go*

Intersecciones, grupos, libertades: Sobre un tablero cuadrado, inicialmente vacío, que tiene marcada una retícula de 19x19 intersecciones se sitúan piedras blancas y negras por turnos, teniendo cada jugador un color. Las piedras se sitúan únicamente sobre intersecciones vacías y no se mueven. Las piedras que tienen una piedra adyacente del mismo color, según las direcciones horizontal y vertical marcadas por la cuadrícula forman un grupo indivisible.

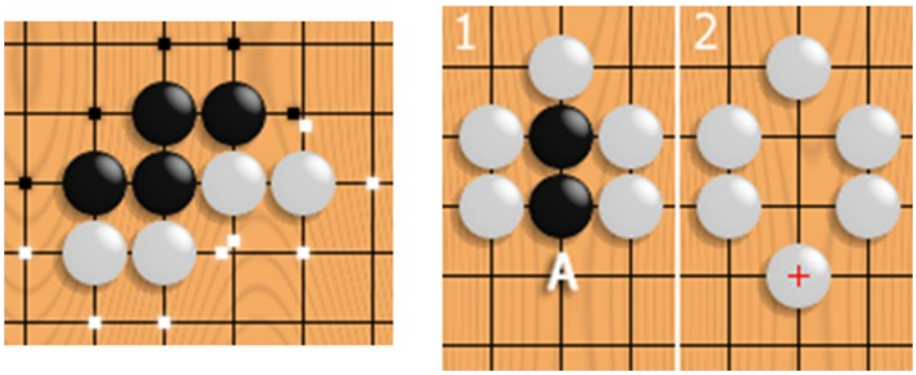

Figura 2.3. Grupos y libertades.

Así, la figura **2.3** a la izquierda muestra tres grupos (uno negro y dos blancos). Los grupos tienen siempre un cierto número de intersecciones vacías alrededor. Solamente se cuentan las intersecciones vacías en sentido horizontal y vertical. Estas intersecciones vacías adyacentes se denominan libertades. En la figura de la izquierda están señaladas las 5 libertades del grupo negro y las 4 de cada uno de los grupos blancos. Obsérvese que una de las libertades del grupo negro es compartida por dos piedras del mismo, pero sigue siendo una única libertad ya que es una única intersección.

Capturas: Cuando un grupo tiene ocupadas todas sus intersecciones adyacentes por piedras enemigas, es decir, cuando pierde su última libertad, es capturado y retirado del tablero. En la figura **2.3** a la derecha vemos (mitad izquierda) un grupo negro que tiene solamente una libertad (la intersección A). Al jugar el blanco sobre este punto (mitad derecha), el grupo negro pierde su última libertad y es retirado del tablero.

Más reglas necesarias: Lo ya expuesto es todo el fundamento del *go*. De estas reglas básicas surge la necesidad de precisar cinco cosas más: la prohibición del suicidio, la regla del *ko*, cuándo termina una partida, el concepto de vida incondicional y el de vida en *seki*. Pero todos estos conceptos no son más que decisiones lógicas impuestas por las situaciones que descubrimos a medida que vamos explorando lo que puede resultar de la aplicación de los fundamentos aquí expuestos. En la introducción en inglés hemos incluido referencias de algunos libros y descripciones [7, 8, 10] que pueden ser consultados para comprender completamente el juego. También el tutorial descrito en [9] está disponible en español.

Objetivo del juego: Los principales objetivos pueden ser:

a. Pierde el primero que no puede poner más piedras. Esto se conoce como **reglas chinas tradicionales** y ya no se juega porque la última fase de la partida carece por completo de interés al ser meramente rellenar de piedras el territorio que se ha obtenido previamente.
b. Gana el que ocupa más "espacio". Definimos "espacio" incluyendo las piedras. Este espacio se conoce como **área** o **reglas chinas**. Es más sencillo para principiantes pues no necesita evaluar si los grupos que quedan están vivos (no pueden ser capturados) y permite realizar jugadas innecesarias sin penalización.
c. Gana el que ocupa más "espacio". Definimos "espacio" contando solamente las intersecciones vacías enteramente rodeadas por un mismo color. Este territorio puede contener grupos "muertos", es decir que pueden ser capturadas por el adversario. Se conoce como **territorio** o **reglas japonesas**. Además, hay que sumar la diferencia entre el número de piedras capturadas por ambos jugadores. Tiene la ventaja de necesitar contar muchos menos puntos. Cuando ya no se es un principiante, resulta más fácil para evaluar la posición durante el juego al contar solamente lo imprescindible.

Un principiante puede crearse la idea falsa de pensar que se trata de objetivos diferentes jugar con reglas chinas o japonesas, en realidad son el mismo objetivo. Es cierto que hay pequeñas diferencias que no merece la pena analizar aquí, pero éstas resultan en una diferencia que casi nunca excede de un punto y solamente deben ser tenidas en cuenta al final de la partida. La evolución del objetivo es una evolución natural que ha tenido el juego a medida que se ha ido profundizando en lo esencial del mismo. El contenido de esta tesis no depende de que se utilicen reglas chinas o japonesas (las reglas chinas tradicionales ya no se utilizan). Como se ha dicho, la diferencia solamente es relevante al final de la partida. Para un programa de ordenador resulta más sencillo

utilizar reglas chinas ya que no requieren analizar qué está vivo y qué está muerto. Si se quiere obtener una aproximación suficientemente buena al valor que producirían las reglas japonesas (en términos de diferencia entre ambos jugadores, no el valor de cada uno) hay que contar el número de veces que juegan (no pasan) los jugadores durante la partida y compensar la diferencia en puntos. El resultado será correcto, excepto en casos de reglamentos específicos.

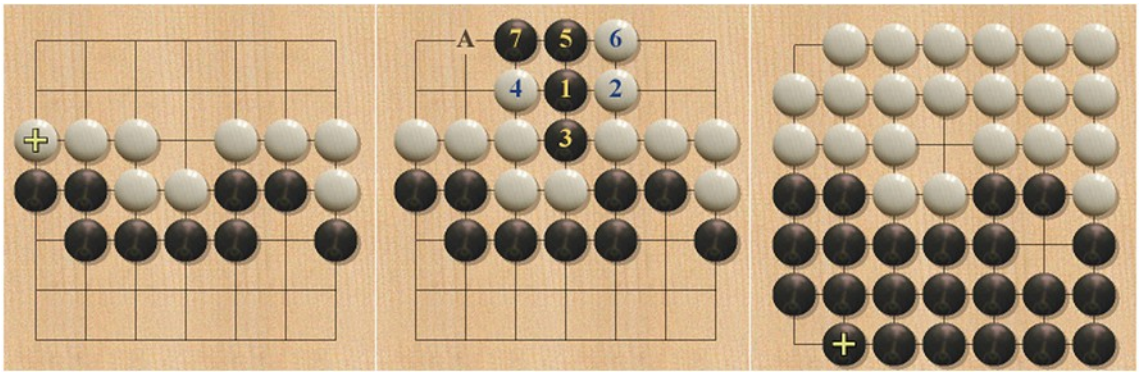

Figura 2.4. Ejemplo con reglas chinas tradicionales.

En la figura **2.4** desde la posición de la izquierda la partida está decidida. La intención de este ejemplo es insistir en que hasta esta jugada ambos jugadores estaban repartiéndose el tablero en jugadas que tenían un valor en puntos, blanco peleaba para extenderse hacia abajo y negro hacia arriba. Un principiante podría pensar que negro (que es el siguiente en jugar) puede invadir la parte superior. En el centro se muestra un intento fallido, la secuencia 1, 2 ... 7 fracasa para negro, cuyo grupo es capturado jugando blanco en A. Realmente, con una respuesta correcta todos los posibles intentos fallarían. Por lo tanto, no queda más remedio (según las reglas chinas tradicionales) que seguir jugando cada uno en su propio territorio hasta alcanzar la posición de la derecha, que deja 2 libertades a cada grupo. Como el último en jugar ha sido negro, éste ganaría la partida ya que blanco no puede jugar. Se ve que incluso con reglas chinas tradicionales el objetivo del juego no es otro que conseguir el mayor territorio seguro posible y el resto de la partida equivale a contar el territorio previamente obtenido. Si negro no hubiese tenido más territorio (1 punto más) no hubiese podido jugar el último o se habría obligado a jugar dejando un solo "ojo" (en este sentido: libertad completamente rodeada de piedras del propio color o rodeada por el borde), lo que habría permitido capturar su grupo. En este caso, aunque hubiera seguido jugando, todas sus piedras habrían podido ser capturadas jugando siempre blanco el último y, por lo tanto, ganando. Igualmente, si en la posición de la derecha, blanco cometiera el error de jugar una piedra más como muestra la figura **2.5** ya solamente tendría una libertad y sería capturado por negro.

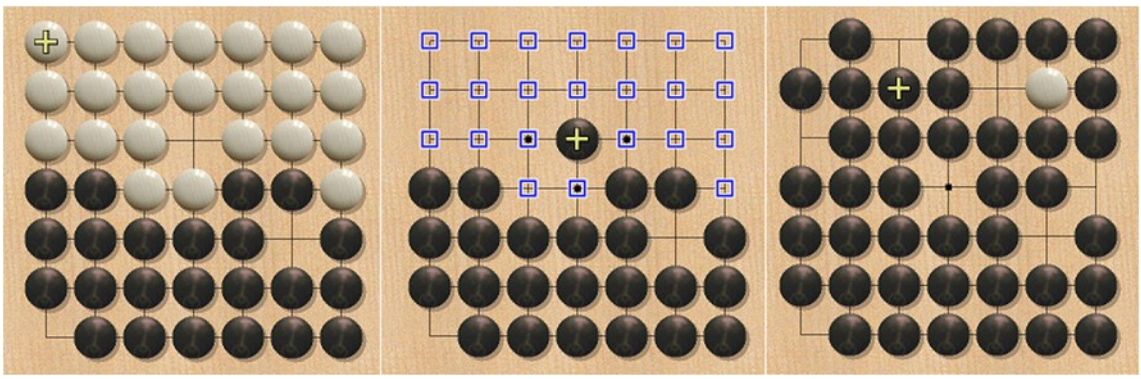

Figura 2.5. Si un grupo no mantiene dos "ojos", puede ser capturado.

Es decir, blanco debe dejar que su grupo tenga dos "ojos" (hablando de reglas chinas tradicionales, los ojos serían de un solo punto no adyacentes). En el juego actual (reglas chinas o japonesas) no se rellenan los territorios de piedras del mismo color, así que no es necesario dejar los ojos explícitamente, es suficiente con que sea evidente que podría jugarse para dejar los dos ojos. Los jugadores experimentados son capaces de saber en posiciones cerradas si pueden conseguirse dos ojos (es decir, el grupo está vivo) o si no (el grupo está muerto). Los jugadores principiantes pueden intentar la captura continuando la partida para determinar si está vivo o finalmente muere. Este añadir jugadas (posiblemente) innecesarias no tiene penalización alguna con reglas chinas, pero sí puede modificar el resultado bajo reglas japonesas. En la figura **2.5** tras la jugada blanca de la izquierda, negro captura (centro). Negro siempre sería el último en jugar porque cualquier piedra blanca sería capturada finalmente tras una breve secuencia hasta alcanzar una posición como la de la derecha, donde blanco ya no tiene ninguna jugada legal. Poner piedras en intersecciones vacías de un solo punto está prohibido, porque se capturarían a sí mismas al no tener libertades. Lo mismo ocurre con la piedra blanca, que no puede extenderse hacia la izquierda porque crearía un grupo sin libertades que se capturaría a sí mismo. Esto se conoce como la prohibición del suicidio.

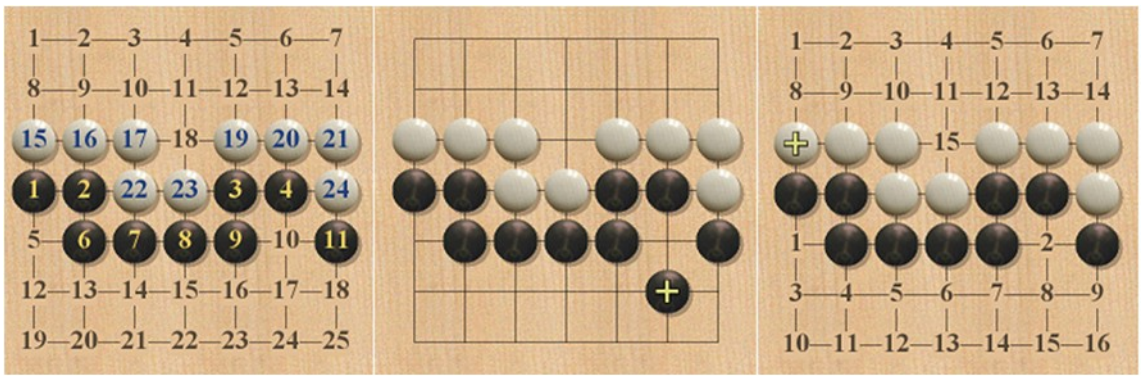

Figura 2.6. Contando el resultado final.

Reglas chinas: En la partida anterior, simplemente cuando ambos jugadores consideran que ya no pueden conseguir más de lo que tienen, ambos pasan y se termina la partida. Se cuenta el territorio rodeado y las propias piedras (figura **2.6** izquierda) resultando, blanco: 24, negro: 25, negro gana por 1. Normalmente, puesto que negro tiene ventaja por empezar la partida se daría a blanco unos puntos de compensación (denominado *komi*). Obsérvese, que si negro fuera inexperto y no supiera si su territorio está seguro (no puede ser invadido con éxito), podría añadir una piedra innecesaria en su propio territorio sin que cambie el marcador (figura **2.6** centro).

Reglas japonesas: En la partida anterior, siempre que no hubiera habido capturas (o ambos bandos hubieran capturado el mismo número de piedras) y, nuevamente, sin compensación de *komi*, el resultado con reglas japonesas es (figura **2.6** derecha): blanco 15, negro 16, negro gana por 1. Sin embargo, si negro hubiera añadido piedras innecesarias dentro de su propio territorio (como en la figura **2.6** centro) perdería un punto por cada una. En la partida de ejemplo sobre un tablero de 7x7 no parece que la diferencia sea grande, pero en una partida real (figura **2.7**) queda claro, que los puntos que hay que contar son muchos menos cuando se utilizan las reglas japonesas.

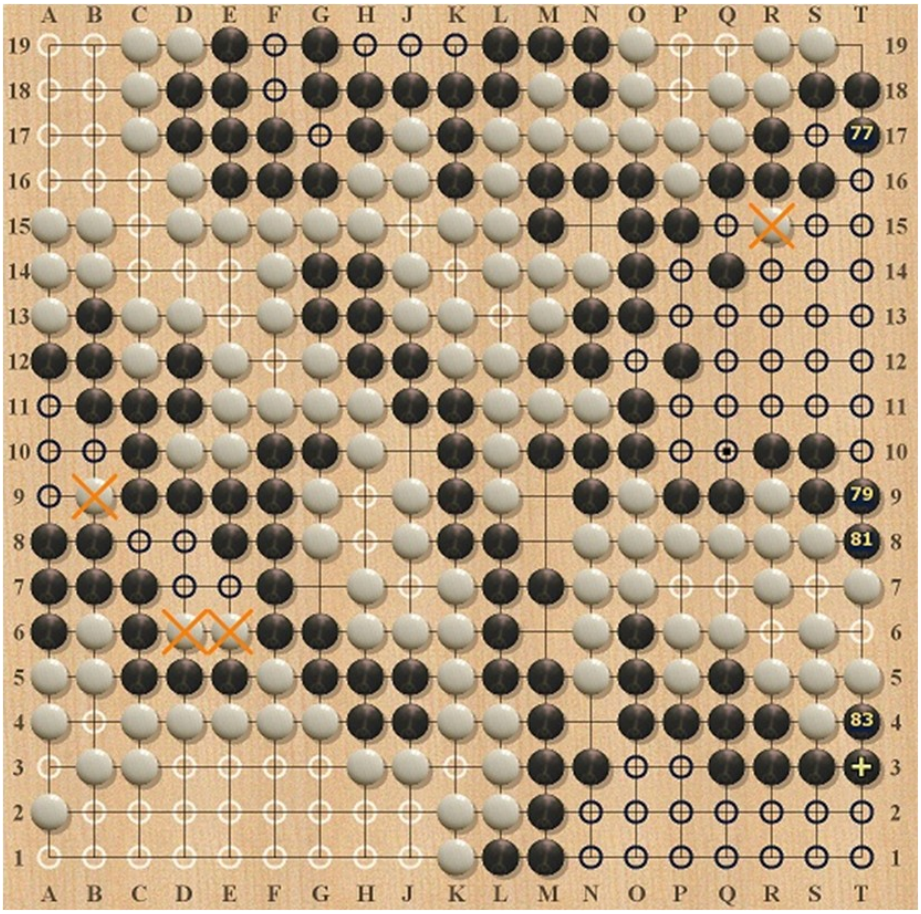

Figura 2.7. Posición final evaluada con reglas japonesas.

Además, los grupos muertos (es decir, que pueden ser capturados) tachados con una X naranja se dejan sobre el tablero y se cuentan como capturas. Los puntos que no tienen ningún círculo (ni negro, ni blanco) no son de nadie. Aunque siguiéramos jugando, nunca podrían llegar a ser territorio.

2.1.3 Evaluación de la habilidad del los jugadores

Para poder comparar el rendimiento de los jugadores de cualquier juego se suele utilizar el índice Elo [28]. Este sistema de clasificación, propuesto en 1939 por el matemático húngaro Arpad Elo se aplicó rápidamente al ajedrez y se ha extendido a los demás juegos. En ajedrez, solamente 4 jugadores en el mundo superan el índice Elo de 2800, 18 más superan el 2700. A partir de 2400 un jugador de ajedrez tiene la categoría de maestro internacional, 2200 es el nivel de un maestro candidato, algo que alcanzan unos 20 000 jugadores en el mundo. En *go*, lo tradicional es utilizar el sistema japonés (*kyu, dan, profesional*) descrito más abajo pero, por supuesto, también se puede medir el rendimiento de los jugadores utilizando el sistema Elo. La hipótesis de partida del sistema Elo es que cada jugador tiene las mismas probabilidades de ganar a su adversario que de extraer un número más alto que éste si ambos extraen un número siguiendo una distribución normal de idéntica desviación típica y cuya media es el índice Elo. La probabilidad de ganar se puede aproximar con la sencilla fórmula basada en la curva logística:

$$Prob_{ganar\ \ A\ a\ B} = \frac{1}{1+10^{(Elo_B - Elo_A)/400}} \qquad (2.1)$$

Sirva de dato, basándose en la clasificación de jugadores humanos utilizando el índice Elo, que todo el conocimiento de ajedrez desde un neófito que lee las reglas por primera vez al mejor jugador del mundo, puede categorizarse en 14 escalones representando cada escalón una desviación típica de la distribución normal antes mencionada. En *go*, la misma escala tiene más de 30 escalones.

El sistema japonés usado habitualmente en *go* utiliza la palabra ***kyu*** (que significa grado) y se utiliza para asignar niveles de juego bajos e intermedios en orden decreciente, un novato es 35 *kyu*, un aficionado que ya domina el juego es 1 *kyu*. A partir de ahí se pasa a ser ***dan*** en orden creciente. 1 *dan* es el nivel más bajo dentro de lo que se puede considerar como "maestro", algo como un cinturón negro de artes marciales. La escala dentro de los "maestros" empezaría en 1 *dan* hasta el máximo de 9 *dan*. Todos los jugadores profesionales son 9 *dan* y entre ellos hay una escala que va de 1p a 9p basada en los títulos que han ganado. (Esto puede variar ligeramente de unas ligas profesionales a otras.) Entre cada grado, se considera que (hasta un máximo de 6) la diferencia de grado puede compensarse dando piedras de ventaja al jugador inferior (que juega con negras) a razón de una por cada nivel de diferencia. El jugador superior no recibe en este caso compensación de ventaja

komi dejándose el valor del *komi* en medio punto para evitar el empate. Así, un 7 *kyu* con 4 piedras de ventaja estaría en igualdad de condiciones frente a un 3 *kyu* y un 2 *dan* con 4 piedras de ventaja frente a un 6 *dan*. Esto no se aplica a los grados profesionales pues las piedras de ventaja representan diferencias demasiado grandes a un nivel de juego "casi perfecto".

2.1.4 El papel del conocimiento en *go*

Para ilustrar cómo al desarrollar las consecuencias que surgen de aplicar un conjunto elemental de reglas, puede surgir una ciencia con una gran cantidad de conocimiento, podemos comparar el *go* con la teoría de números. Igual que la teoría de números se fundamenta en una única operación: contar, es decir, definir $succ()$, el siguiente de cada número. Con una definición recursiva basada en $succ()$, podemos definir la suma, con una definición recursiva basada en la suma, la multiplicación, definimos las inversas y … El edificio de la teoría de números surge naturalmente de una operación tan elemental. De las reglas aquí expuestas surge el edificio del *go*, el juego de estrategia más profundo conocido.

Además, el *go* es un problema combinatoriamente intratable. Al ser un juego no aleatorizado, bipersonal, finito, de suma cero e información completa, cumple las condiciones del teorema minimax. Esto implica que toda posición tiene un valor minimax. Excepto para posiciones triviales de finales de partida, este valor es imposible de computar. La complejidad del espacio de estados ha sido estimada [29] en 10^{171} (frente a 10^{50} del ajedrez) y la complejidad del árbol en 10^{360} (frente a 10^{123} del ajedrez). Ni siquiera se conoce una función de evaluación determinista que permita construir un programa basado en búsqueda minimax global que sea mejor que un humano principiante. A diferencia del ajedrez donde una función tan elemental como la movilidad (el número de jugadas legales que hay en una posición) y una búsqueda minimax producen un programa que supera los 2000 puntos de índice Elo sobre un ordenador doméstico actual [19], algo que pocos jugadores humanos alcanzan en su vida y quienes lo hacen lo consiguen con gran esfuerzo.

El *go* no es solamente un problema intratable. Del estudio del juego realizado durante siglos surgen conceptos tácticos y estratégicos que permiten analizarlo de forma estructurada. Cualquier jugador de *go* conoce unas 50 palabras, la mayoría en japonés que dan nombre a situaciones y permiten analizar el papel de la iniciativa, de la forma de los grupos, de las múltiples alternativas para conseguir un mismo objetivo, de las interacciones entre objetivos, etc. Hay situaciones cuya evaluación correcta exige anticipar decenas de jugadas más o menos forzadas y que un jugador entrenado es capaz de evaluar mediante un análisis estático. Este "lenguaje del *go*" constituye un marco en el que estudiar el valor y la urgencia de cada jugada posible. El *go* es

interesante no tanto por ser intratable, como por ser un problema en el que se necesita de inteligencia para avanzar, el competidor humano ha alcanzado niveles de excelencia y es evaluable de manera no antropocéntrica (ya que se trata de un juego en el que se gana o se pierde) a diferencia de otras actividades intelectuales humanas donde la evaluación está sesgada por antropocentrismo.

2.2 El *go* como problema computacional

El primer programa que juega al *go* fue escrito por Albert Zobrist en 1968. Posteriormente, creó además una función para representar el tablero mediante un identificador entero, conocida como *Zobrist hashing* [30] que sigue siendo ampliamente utilizada por su eficiencia, sencillez y capacidad para ser actualizada incrementalmente. La idea es muy sencilla, se trata de construir una tabla de números enteros aleatorios (generalmente de 64 bits) para cada intersección del tablero si en ella hay una piedra blanca, una segunda tabla del mismo tamaño para las piedras negras, una tercera para indicar si está prohibido jugar en una intersección porque se produciría la repetición de una posición anterior (regla del *ko*) y un número aleatorio para indicar si juega blanco o negro. El hash que representa la situación, posición de todas las piedras, qué jugador tiene el turno y si hay alguna intersección en la que esté prohibido jugar es la O-exclusiva bit a bit de todos los números que definen la posición. La posición se identifica con un solo número de 64 bits, con lo que es muy fácil ver si ya ha sido evaluada previamente (algo muy frecuente en cualquier tipo de búsqueda de juegos), actualizar dicho hash es tan sencillo como realizar una O-exclusiva de los elementos que hayan cambiado y, debido a la aleatoriedad de los números, la probabilidad de una colisión es de 2^{-64} para un *hash* es de 64 bits.

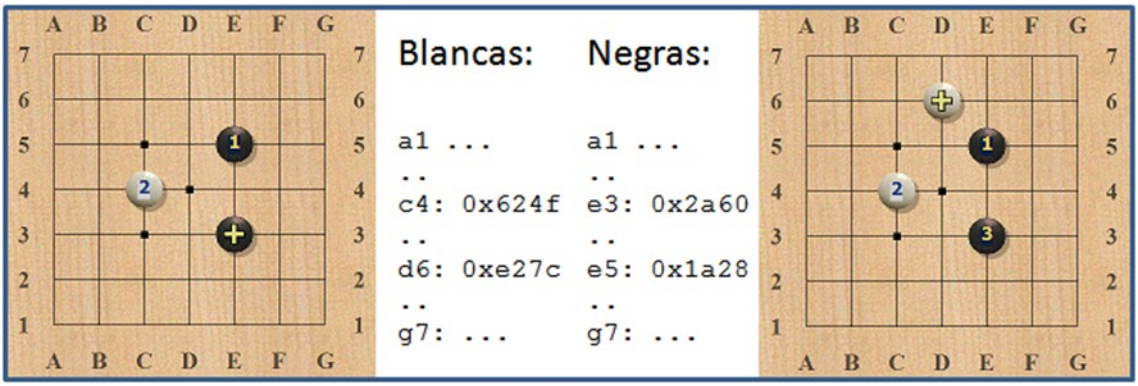

Figura 2.8. Cálculo del *hashing* de Zobrist de una posición.

En el ejemplo de la figura **2.8** aparecen en el centro los valores relevantes de la tabla de números aleatorios para cada una de las intersecciones del tablero. La posición de la izquierda tendrá como hash el valor

e3 y e5 negras con c4 blanca: 0x2a60 xor 0x1a28 xor 0x624f = 0x5207. La posición de la derecha tendrá ese mismo valor con el añadido de una piedra blanca en d6: 0x5207 xor 0xe27c = 0xB07B. Nótese que obtenemos el valor de la posición de la derecha incrementalmente a partir del anterior y no volviéndolo a calcular desde un tablero vacío. También funciona "deshacer" una jugada volviendo a obtener el hash de la izquierda a partir del de la derecha.

Un trabajo de investigación pionero destacable, que es utilizado en lo que se conoce como "programas clásicos", es el algoritmo de vida incondicional de David Benson. Durante 30 años (1976-2005) el algoritmo de Benson [31] estaba en la evaluación final de los programas de *go* cuando éstos necesitan determinar el estado de un grupo. El problema de la vida incondicional es determinar si un grupo puede ser capturado o no por un adversario que puede jugar ilimitadamente. Por ejemplo, los 2 grupos de la figura **2.4** (derecha) están incondicionalmente vivos. En cambio, los grupos de la de la figura **2.4** (izquierda) no lo están, están vivos (no pueden ser capturados frente a una defensa correcta), pero no incondicionalmente vivos (pueden ser capturados si el adversario puede jugar ilimitadamente sin que haya defensa). Si bien, en una partida real pocos grupos llegan a estar incondicionalmente vivos, el algoritmo es muy importante, pues representa el final de la búsqueda minimax. Los grupos no incondicionalmente vivos pueden reducirse mediante una secuencia (que encontrará la búsqueda minimax) en incondicionalmente vivos o, si no, es que no están vivos. Hasta Benson, determinar la vida de un grupo se realizaba mediante un análisis hacia adelante (*look ahead*) hasta que se capturaba (o fracasaba en capturar) el grupo. Benson construye por primera vez un lenguaje matemático del *go* y utiliza la teoría de grafos para demostrar que su análisis estático es equivalente a la búsqueda dentro de un marco formal de definiciones, teoremas y demostraciones. En el periodo (1976-2005), se publican más de 250 artículos sobre *go* por ordenador, decenas de libros y tesis doctorales. Una lista incompleta de más de 300 artículos (hasta la actualidad) puede verse en la Universidad de Alberta [18]. Empiezan las competiciones entre programas e incluso un millonario de Taiwán creó un premio para evaluar el progreso [32].

Los mejores programas clásicos que aún existen, algunos de los cuales siguen haciendo nuevas versiones, alcanzaron un nivel de juego equivalente a 7 *kyu* siguiendo la escala descrita en 2.1.3.

Por lo tanto, un programa 7 *kyu* estaría en igualdad de condiciones recibiendo 7 piedras de ventaja de un jugador que fuese 1 *dan*. Es decir, el programa puede recibir 7 piedras de ventaja de una persona que a su vez puede recibir 9 piedras de ventaja de un profesional.

En 2006 los matemáticos húngaros Levente Kocsis y Csaba Szepesvari publican un trabajo que abre una nueva vía [20]. Inicialmente, los programas que utilizan el algoritmo se conocen como UCT porque la función

propuesta en el trabajo original formaliza el problema en términos de un intervalo de confianza (*Upper Confidence-bound for Trees*), pero el algoritmo general es rápidamente modificado para combinarlo con conocimiento específico y los distintos autores pasan progresivamente a utilizar el término más genérico de *Monte-Carlo Tree Search* (MCTS) para incluir todos los algoritmos basados en una búsqueda mediante simulación que organiza los resultados obtenidos en un árbol.

MCTS combinado con diferentes heurísticas y formas de integrar conocimiento a priori permite avanzar el *go* por ordenador hasta el nivel de 4 o 5 *dan*. Es decir, programas que están equilibrados recibiendo 4 o 5 piedras de ventaja de un jugador profesional. Siendo este avance realizado en seis años el salto más grande en la historia del *go* por ordenador. La distancia con el juego profesional es aún enorme ya que, a medida que el nivel de juego se aproxima a la perfección, una piedra de ventaja representa una diferencia mayor, ya que las partidas se deciden por "sutilezas" evaluadas con gran precisión, mientras que para un novato una piedra de ventaja apenas representa nada.

UCT (o MCTS) ha sido aplicado con éxito a diversos problemas de uno o dos jugadores. En un trabajo del año 2009 [33] comunicamos los resultados de una aplicación al *Strip Packing Problem* (SPP) y igualando los mejores rendimientos publicados entonces en SPP obtenidos por programas que tenían un alto nivel de heurísticas especificas del problema, mientras que el programa basado en UCT es un programa carente por completo de conocimiento del problema. Incluso encontró un empaquetamiento óptimo (ver figura **2.13**), algo que no consiguió el programa con el que se estaba comparando. En esta tesis se describe una aplicación en genética humana en el capítulo 5. También ha revolucionado el mundo de *General Game Playing* [34, 35], una competición en la que los programas compiten en un juego cuyas reglas les son desconocidas hasta el momento de la competición. Los programas aprenden el juego leyendo una definición formal del mismo escrita en GDL [36] (*Game Description Language*). La lista de aplicaciones actualizada a 2012 puede verse en 7.1.

El mundo académico, especialmente el anglosajón, pero también en los principales países europeos y en Asia ha tenido siempre la mirada puesta en los avances en computer *go* formando equipos, organizando y participando en competiciones y publicando. En el mundo anglosajón, no solamente el mundo académico sino también los medios generales han recogido los principales avances en computer *go*. Por ejemplo: *The Times, New York Times, The Economist, Reuters, Abc News, Scientific American, The Guardian, Slashdot, Wired*, etc.

2.3 Introducción a MCTS

Este apartado contiene una descripción informal sobre las razones que justifican MCTS. Pretende servir para entender cuáles son sus puntos fuertes y sus limitaciones, haciendo una descripción de las ideas que llevaron a MCTS. Se desarrolla también un ejemplo aplicado a un problema de optimización, el SPP (*Strip Packing Problem*). Para comprender cómo se incorpora MCTS en un programa de *go* competitivo, hay que ir a los capítulos 3 y 4.

2.3.1 UCB y el dilema de exploración/explotación

UCT (*Upper Confidence-bound for Trees*) es un algoritmo que incorpora un algoritmo anterior llamado UCB (*Upper Confidence Bound*) y lo extiende a un árbol. UCB fue creado para abordar el problema del "*Single-armed Bandit*" [37], en particular, el dilema de exploración/explotación incluido en el mismo. El problema del "*Single-armed Bandit*" ("bandidos de un sólo brazo" es una expresión popular inglesa para referirse a las máquinas tragaperras) consiste en encontrar la estrategia óptima que tiene un jugador para maximizar sus ganancias cuando juega un número n de tiradas de un conjunto de k máquinas que distribuyen un premio según la uniforme $U(0, p_i)$ $i \epsilon \{1, .. k\}$ donde p_i es desconocido y debe estimarse a partir de los resultados que el jugador va obteniendo. El jugador puede buscar dos estrategias extremas:

a. Jugar una vez en cada máquina y, a partir de ahí, jugar solamente en la máquina en la de la que obtuvo el premio más alto.
b. Jugar en todas las máquinas por igual.

Es importante destacar que lo que el jugador persigue es únicamente maximizar su ganancia y no la información sobre la p_i de cada máquina. El jugador pierde beneficios potenciales si estima erróneamente cuál es la mejor máquina, pero no pierde absolutamente nada por estimar erróneamente el valor exacto de la p_i de una máquina que es "de las peores". Es decir, una máquina de la que, tras un pequeño número de pruebas, tenemos un nivel de certeza razonable de que no puede ser la mejor.

La estrategia **a** corresponde con la máxima **explotación** y la estrategia **b** con la máxima **exploración**. Se entiende que ninguna de las dos es óptima y que la estrategia óptima tiene que ver con el nivel de confianza que tengamos sobre cuál es la mejor máquina. Es decir, con los límites de un intervalo para un cierto nivel de confianza predeterminado. Por lo tanto, el dilema de exploración/explotación consiste en encontrar el balance adecuado entre jugar más veces nuestra mejor máquina (para obtener mayor beneficio) y explorar las mejores candidatas alternativas (para estar seguros de que no nos estamos equivocando de máquina).

2.3.2 Los métodos Monte-Carlo ingenuos

Puesto que en *go* el espacio de búsqueda es prácticamente infinito y no existe una función de evaluación fiable, excepto para las posiciones finales del juego, podría proponerse como método:

1. Tras una jugada inicial m_i jugar partidas de *go* aleatoriamente hasta que no se puedan poner más piedras siguiendo una cierta heurística.
2. Como esas posiciones finales sí son fáciles de evaluar, anotar cuántas victorias y derrotas produce cada jugada inicial m_i repitiendo el proceso un cierto número de veces n para cada una de las I jugadas legales iniciales $i \in \{1, .. I\}$.

Es decir, de ese espacio de búsqueda prácticamente infinito tomamos una muestra aleatoria, la evaluamos y utilizamos métodos estadísticos para generalizar lo que hemos aprendido de una muestra a toda la población. A una partida jugada aleatoriamente la denominamos una simulación y nuestra función de evaluación será: condicionado a que juguemos cada jugada legal, qué proporción de simulaciones ganamos. De esta manera, surgen los métodos Monte-Carlo ingenuos (inglés *naif*) que alcanzan un nivel de juego parecido al de un principiante o un poco mejor para tamaños de tablero pequeños. La "ingenuidad" de estos métodos está sobre todo en que consideran, dentro de las simulaciones, todas las jugadas por igual, es decir, no tienen en cuenta que ciertas respuestas son obvias, incluso para un principiante.

2.3.3 UCT es UCB aplicado en un árbol a una evaluación estocástica

Tabla 2.1. Similitud entre el problema UCB y los métodos Monte-Carlo.

UCB	MÉTODOS MONTE-CARLO	DESCRIPCIÓN
n (número de jugadas)	**n** (número de simulaciones)	Es un coste. La capacidad de cálculo es finita. Queremos identificar la mejor máquina/jugada pudiendo realizar un cierto número de ensayos.
Ganancia de una tirada	Resultado de una simulación	No tenemos otra función de evaluación que el resultado de un experimento aleatorio. Función de **evaluación estocástica**.
Explotación: Maximizar las ganancias = jugar más veces la mejor máquina. **Exploración**: para estar así razonablemente seguros de saber cuál es la mejor máquina ya que la evaluación tiene ruido.	Podemos simular cada jugada el mismo número de veces (máxima exploración) o aplicar UCB y conseguir una estimación de menor varianza para el número de victorias de las mejores jugadas.	El balance exploración/explotación es controlable mediante UCB

En la tabla 2.1 se resumen y muestran las analogías que existen entre el algoritmo UCB y los métodos Monte-Carlo, vemos que ambas técnicas son similares. La primera dificultad es que la partida uniformemente aleatoria no representa una partida razonable, porque no siguen a cada jugada las continuaciones esperadas. No sirve de nada realizar una jugada que obtendría un gran beneficio frente al 99% de las respuestas posibles (y, por lo tanto, parecería muy buena si el juego recibe una evaluación uniformemente aleatoria) y, en cambio, resulta una mala jugada si el adversario contesta correctamente a la misma. Además, a menudo esta contestación correcta será obvia para un jugador humano. Esto no es tenido en cuenta en las simulaciones y es una gran limitación, precisamente la idea central de la búsqueda minimax es cubrirse frente a las mejores respuestas del adversario, no frente a respuestas aleatorias.

La solución que ofrece MCTS es construir un árbol en memoria con el conocimiento del que disponemos (los resultados de las simulaciones). La exploración dentro del árbol deja de ser aleatoria y favorece la explotación, es decir, recorrer más veces las mejores ramas del árbol, incluyendo las mejores respuestas a las jugadas. Es muy importante porque es el árbol el que organiza las jugadas y sus respuestas. De esta forma, a cada jugada le siguen (suponiendo que se hayan realizado suficientes simulaciones) las respuestas más probables mejorando la evaluación.

UCT (la primera forma de MCTS) consiste en quedarnos con lo mejor de los dos mundos construyendo un árbol en memoria. Cada iteración del algoritmo recorre el árbol desde el nodo inicial siguiendo en cada nodo el camino que indique la evaluación UCB hasta llegar a una hoja del mismo. Alcanzada esa hoja, realizaremos una simulación a partir de la misma. Conocido el resultado de la simulación, lo propagaremos hacia atrás por el árbol para actualizar el conocimiento del problema. Además, habrá que tener un criterio para decidir cómo crece el árbol. El criterio normal es extender las hojas que han sido visitadas un cierto número de veces, esto consigue que el árbol crezca más en las direcciones más prometedoras de manera automática aprovechando el propio UCB.

La evaluación UCB consiste en elegir en cada nodo j aquél hijo i que maximice:

$$V = \frac{v_i}{n_i} + K \times \sqrt{\ln(n_j) / n_i} \qquad (2.2)$$

Donde:

v_i es el número de victorias acumuladas por las simulaciones del nodo hijo i

n_i es el número simulaciones que han pasado por el nodo hijo i

n_j es el número simulaciones que han pasado por el nodo padre j

K es una constante que permite controlar el balance exploración/explotación.

La ecuación 2.2 es la propuesta originalmente por Kocsis y Szepesvari [20]. Se puede observar que, para un valor dado del número de visitas del padre (es decir, a la hora de comparar unos hijos con otros), la ecuación no representa otra cosa que la proporción de éxitos observados (el valor máximo verosímil en la estimación) más un intervalo de confianza basado en la normal (cuya varianza disminuye como $1/\sqrt{n}$) para una cierta significación (cada valor de significación se corresponderá con un cierto valor de K). Tiene sentido usar la distribución normal como la hipótesis más sencilla y porque la repetición de un experimento en idénticas condiciones se distribuye como una binomial, que se aproxima mediante la normal tras un número suficiente de observaciones.

2.3.4 Ejemplo: UCT aplicado al SPP

Podemos ilustrar UCT con un ejemplo tomado del *Strip Packing Problem*. Este problema es un caso particular de los problemas de empaquetamiento (*Packing Problems*). Según la clasificación comúnmente aceptada [38], se denomina el problema de Strip Packing 2-D ortogonal. Se trata de empaquetar un conjunto previamente conocido de rectángulos de manera que ocupen el menor espacio posible. Los rectángulos pueden rotarse 90 grados. Hay que situarlos sobre una cinta cuya anchura es un dato del problema. Por lo tanto, minimizar la longitud de cinta empleada equivale a minimizar el área utilizada. Consideramos que el problema de colocar rectángulos se reduce a elegir entre jugadas legales de un juego. Únicamente consideramos colocar un rectángulo sobre la línea más baja de la cinta, si hubiera más de una, la que esté más a la izquierda. Y, dentro de esta línea, coloco el rectángulo a la izquierda. Esta simplificación del problema se conoce como heurística *bottom-left* (BL). De esta forma, el conjunto de jugadas legales queda definido por la longitud del segmento más bajo de la cinta y el conjunto de los rectángulos que queden por colocar. Una simulación consistirá en elegir aleatoriamente entre los rectángulos que puedan colocarse en el segmento BL y sus orientaciones (0 o 90^0) hasta haberlos colocado todos. Cada simulación tendrá un resultado que es la longitud de cinta utilizada que queremos minimizar.

Supongamos que tenemos 4 rectángulos que denominamos A, B, C y D y que los podemos colocar también rotados lo que indicaremos con A', B', C', D'

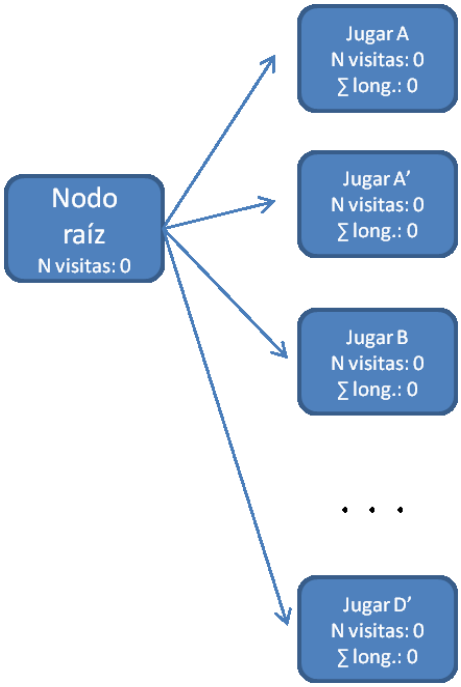

Figura 2.9. El nodo raíz desplegado antes de la primera simulación.

Inicialmente, el árbol no está creado, solamente tenemos un nodo raíz vacío. Desplegamos el nodo raíz añadiéndole un hijo por cada jugada legal (figura **2.9**). Cada nodo acumula el número de veces que ha sido visitado y la suma de los valores objetivos de las soluciones obtenidas que han pasado por el mismo.

La primera simulación, al ver que el nodo raíz está vacío lo expande, a continuación evalúa cuál es el mejor candidato según la función UCB. Evaluamos todos los hijos del nodo actual según:

$$V = \frac{\sum long_i}{n_i} - K \times \sqrt{\ln(n_j) / n_i} \qquad (2.3)$$

Que es la ecuación 2.2 con el intervalo de confianza cambiado de signo ya que en este caso, pretendemos minimizar y en cada nodo, el lugar del número de victorias contamos la suma de longitudes obtenidas $\sum long_i$. Como todos los nodos aún tienen $n_i = 0$, asignamos $V_i = 0$ para que cada nodo reciba al menos una visita. De todos los nodos, nos quedamos con el de menor valor. Al ser todos inicialmente iguales, supongamos que hemos elegido A.

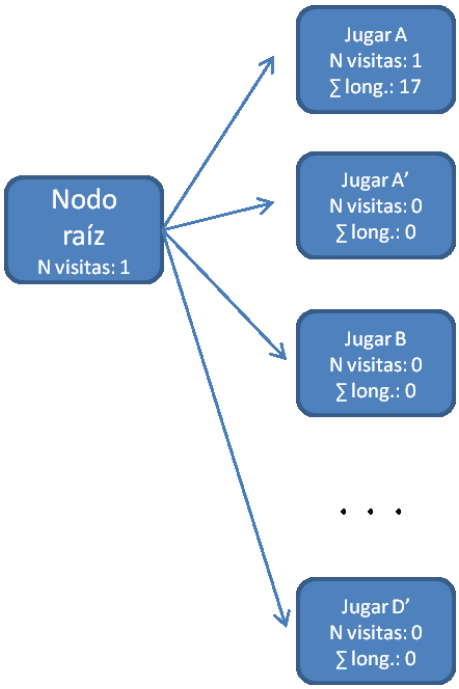

Figura 2.10. El árbol tras la primera exploración del nodo A actualizada.

Por lo tanto, la primera iteración del algoritmo elije el nodo A, coloca sobre la cinta el rectángulo A. Al ver que el nodo A no tiene suficientes visitas, no lo expande todavía. Después de colocar A coloca todos los demás rectángulos en una secuencia aleatoria, supongamos que la secuencia ha sido A, D', B, C y que la longitud de la cinta (valor objetivo de dicha solución) es 17. Ahora propaga el resultado desde el nodo donde terminó la exploración del árbol hasta arriba, quedando como vemos en la figura **2.10**.

Las siguientes iteraciones seguirán eligiendo en primer lugar los nodos que aún no han sido visitados. Supongamos que tras 25 iteraciones el resultado sea el que muestra la figura **2.11**.

Si hemos fijado el umbral para desplegar los nodos en 4 visitas y, suponiendo que el término de exploración $K \times \sqrt{\ln(n_j) / n_i}$ sea pequeño, la iteración 26 elegirá el nodo A', pues $59/4 < 54/3 < 41/2 < 35/1$. Al ver que tiene 4 visitas, lo expandirá, supongamos que tras colocar A' solamente podemos colocar C', D y D' pues el espacio que deja A no es suficiente para B, ni B' ni C. (La aplicación de SPP real tiene una opción adicional, que es "pasar" e ir al siguiente tramo horizontal, que omitimos para simplificar.)

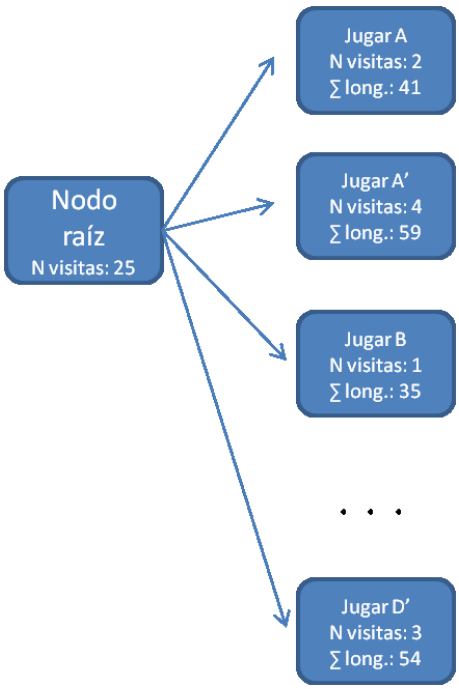

Figura 2.11. El árbol tras las 25 primeras iteraciones.

El nuevo árbol, tras la expansión del nodo, será el que muestra la figura **2.12**.

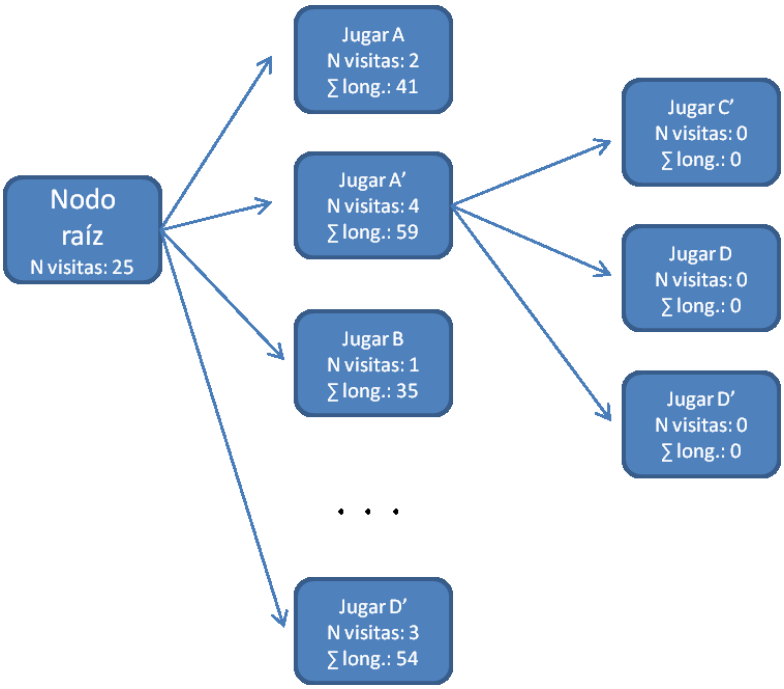

Figura 2.12. El árbol tras la primera expansión de un nodo.

Sobre el nuevo árbol, la parte determinista del algoritmo habrá elegido A', C' y a continuación, una simulación aleatoria elegirá, supongamos B, D. Evaluamos la secuencia A', C', B, D y propagamos el resultado hacia arriba igual que siempre.

Regla de parada: En el caso de *go*, no se necesita regla de parada. Simplemente se realizan tantas iteraciones como sea posible dentro del tiempo disponible y se elije la mejor jugada entre los nodos hijos del nodo raíz (por seguridad, la que más visitas tiene ya que la de mayor valor podría ser exploratoria). En el caso del SPP, el programa en todas las iteraciones compara si la longitud resultante es la mejor obtenida hasta el momento, si es así, almacena la secuencia que la originó. Hay que destacar que una de las ventajas es que el algoritmo puede interrumpirse en cualquier momento y siempre tiene una solución aceptable. En el caso del SPP, también puede interrumpirse el algoritmo cuando la longitud del árbol alcance todas la piezas colocadas ya que a partir de ahí, desaparece el componente aleatorio de la simulación final en la variante principal, que es la secuencia más larga del árbol y la más veces seleccionada.

Convergencia hacia el valor óptimo: Suponiendo que no hubiera limitaciones de memoria ni de tiempo (en el caso de 4 rectángulos, casi inmediatamente), en el caso de una búsqueda minimax, el algoritmo acabaría por tener un árbol que cubriría todo el espacio de búsqueda y que carecería de componente aleatoria final, así que la variante principal seguiría el camino óptimo. Esto no es automáticamente cierto cuando se aplica MCTS a problemas de optimización como el descrito en este ejemplo tal y como se discute en 7.2.6.

Componente de exploración: Si nos fijamos en la expresión $K \times \sqrt{\ln(n_j) / n_i}$ observamos que cada vez que un nodo no es seleccionado, K y su número de visitas n_i son constantes mientras que el número de visitas de su padre, n_j se incrementa en 1. Más o menos lentamente, dependiendo del valor de la constante K, este término aumenta y el nodo acaba por recibir una nueva visita. Si el valor resultante de esa iteración confirma que el nodo tiene un valor $\frac{\sum long_i}{n_i}$ poco interesante, al aumentar n_i, vuelve a disminuir el término y el nodo ya no es seleccionado hasta que aumente más el número de visitas de su padre. Por el contrario, si la evaluación del nodo no era correcta y resulta que su valor disminuye, el nodo pasa a ser seleccionado más fácilmente y recibe más visitas.

Es importante destacar que, a medida que aumenta el número de iteraciones, el valor de cada nodo varía, porque el árbol que le sigue relaciona las secuencias correctamente al ir ganando en conocimiento del problema.

2.3.5 MCTS

Ésta es la versión más simple del algoritmo, conocida como UCT aplicada en un problema de optimización aunque la mayor parte de las aplicaciones descritas son búsquedas de dos jugadores. Por supuesto, en el caso de juegos bipersonales, las victorias se consideran vistas por el jugador que juega en ese nodo. Cada jugador busca maximizar sus victorias que son las derrotas del otro. Por lo tanto, es una forma de búsqueda minimax. También puede usarse, como en el ejemplo anterior, con un único "jugador" para resolver problemas de optimización combinatoria. También se puede considerar un objetivo $r \epsilon R$ (por ejemplo, proporción aprovechada de una lámina) en lugar de $v \epsilon \{0,1\}$ (derrota/victoria) para utilizar el mismo algoritmo en problemas de optimización.

Una gran ventaja es que el algoritmo aprende solo. No necesita de heurísticas introducidas por un experto para saber qué caminos explorar. Pero también puede combinarse con conocimiento a priori para ordenar la búsqueda en problemas muy complejos. Este es precisamente el objeto de esta tesis. A pesar de comprobar que el algoritmo básico es suficientemente bueno para competir con algoritmos especializados en problemas sencillos como el SPP [33] en problemas realmente complicados con una explosión combinatoria intratable incluso para conjuntos de pocas decisiones, el conocimiento del problema aprendido durante la búsqueda u offline resulta imprescindible para proporcionar soluciones competitivas a través de la exploración guiada (capítulo 3) y el aumento de la plausibilidad en las simulaciones (capítulo 4).

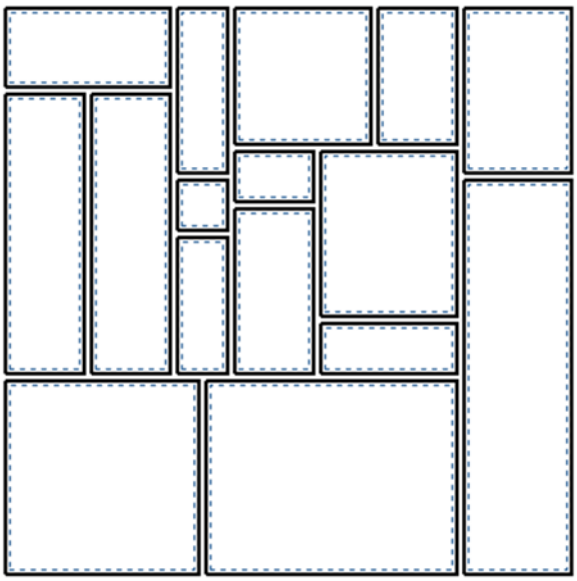

Figura 2.13. Empaquetamiento óptimo del SPP obtenido mediante UCT.

El conjunto de todos los algoritmos que incorporan esta idea, la de organizar en un árbol los resultados de un proceso estocástico (la simulación de una solución elegida al azar) se engloba bajo el nombre de MCTS *Monte-Carlo Tree Search*. UCT es por lo tanto, la forma más sencilla de MCTS utilizando la ecuación 2.2. MCTS a menudo combina la información con otra información a priori, al menos en los programas de *go*.

2.4 Aplicación de MCTS en genética humana

Este apartado resume la aplicación del algoritmo MCTS al campo de la genética humana realizada en esta tesis y descrita extensamente en el capítulo 5.

La aplicación se engloba dentro de lo que se conoce como estudios de asociación *genotipo-fenotipo*.

Un *fenotipo* es un rasgo visible en un individuo. En nuestro caso, una variable categórica.

Para entender lo que es un *genotipo*, se recomienda la lectura del apartado 5.1. Brevemente, el *genoma* es el conjunto de toda la información genética de una especie. Se construye a partir de las secuencias medidas en un conjunto de individuos. Cada individuo tiene variaciones en su genoma con respecto al genoma *de referencia* que es la versión más frecuente en toda la población. Estas variaciones, conocidas como *polimorfismos*, ocupan posiciones fijas conocidas. Existen más de 4 millones de posiciones que se sabe que son variables en el genoma humano. Dentro de estas variaciones, hay un tipo especialmente importante tanto por su frecuencia como por su mayor facilidad de análisis, que es el Polimorfismo de un Solo Nucleótido *SNP* (pronunciado, *snip*). Es decir, un polimorfismo que tiene solamente dos posibles formas para un punto concreto del genoma. Un *alelo* es cada una de las formas de un polimorfismo. En el caso de un SNP cada una de las dos formas: la de referencia y una posible alternativa. Un *genotipo* es un conjunto cualquiera de alelos para los polimorfismos de un individuo. Utilizando técnicas de *microarray*, puede medirse una gran cantidad de SNPs de un individuo en un solo análisis, en nuestro estudio 650 000 SNPs para cada individuo.

2.4.1 Breve descripción del problema

Por lo tanto, un estudio de asociación *genotipo-fenotipo* consiste en clasificar individuos a partir de un conjunto de SNPs de tamaño n. El *fenotipo* puede ser: tener o no tener una enfermedad (en terminología de estudios de casos y controles: ser un caso o un control), un rasgo visible como el color del pelo, etc. Los rasgos

continuos como la estatura, también pueden categorizarse. El apartado 5.1 describe algunos de los posibles mecanismos biológicos que explican los fenotipos. Se comprende que el proceso es muy complejo para que todas las posibles interacciones sean representadas por modelos matemáticos ya que esto obligaría a utilizar modelos con un alto número de grados de libertad, aumentando el problema del *overfitting*. Incluso para un solo SNP, tenemos 650 000 candidatos, lo que obligaría, debido a la existencia de test múltiples a elevar mucho los umbrales de significación utilizando métodos como el de Bonferroni. Sin embrago, ni los tamaños muestrales ni el tamaño de los efectos son suficientemente grandes para utilizar el método de Bonferroni que, claramente, no fue concebido para "millones de tests múltiples". Estos problemas y otros de tipo metodológico recogidos en la tabla 5.11 son descritos en el apartado 5.2.

2.4.2 El algoritmo de clasificación utilizado

Para evitar este *overfitting*, decidimos emplear un método no paramétrico que tiene en cuenta únicamente las distribuciones de frecuencias de cada alelo en cada uno de los grupos del fenotipo. El método compara la *verosimilitud* de que este genotipo haya sido extraído al azar de un sorteo en el que las probabilidades de extraer cada uno de sus alelos son las propias de cada uno de los grupos. Esto es posible porque la frecuencia de cada uno de esos alelos en cada uno de los grupos es conocida. El algoritmo está descrito completamente en 5.5.2. Al no tener parámetros que ajustar, es de esperar que sea más robusto frente a overfitting que otro algoritmo en el que se puedan encontrar valores de parámetros que permitan aprenderse la muestra, pero cuyas conclusiones van a generalizarse mal. Es importante que, además, el algoritmo tenga validación cruzada, en este caso repetida varias veces. Utilizamos las frecuencias aprendidas en un grupo de "entrenamiento" y medimos el número de individuos correctamente clasificados de un conjunto de test. Esto se realiza un cierto número de veces.

2.4.3 Las medidas de asociación comúnmente empleadas

En nuestro primer trabajo publicado en genética [5] implementamos los diferentes métodos de asociación entre un SNP y un fenotipo categórico descritos en 5.3. Tradicionalmente, una vez que se han identificado SNPs cuyo poder de clasificación (frecuentemente, en el caso de enfermedades, hablaremos de riesgo en lugar de clasificación) está probado, la forma de realizar asociación entre varios SNP y un fenotipo es contar el número factores de riesgo. Es decir, un modelo aditivo que suma algo por cada factor de riesgo. Esto, a pesar de haber sido aceptado ampliamente porque la enorme complejidad de buscar modelos de varios SNPs entre

conjuntos tan grandes solamente está siendo abordada recientemente, es incorrecto por varios motivos, incluyendo:

- No representa la complejidad biológica. Las interacciones biológicas son mucho más complejas que una simple suma.
- No representa la asociación estadística entre factores. Muchas veces hay redundancia y un segundo factor apenas aumenta la información que ya tenemos por el primero.

En nuestro estudio probamos experimentalmente que se pueden encontrar conjuntos de SNPs con mucho mayor poder de clasificación que simplemente utilizando los que mejor funcionan individualmente. Esto es además cada día más aceptado y ya ha sido descrito por otros autores, incluyendo [39].

2.4.4 Los experimentos realizados

Los experimentos comparan tres métodos para encontrar conjuntos de $n \in \{4, 8, 12\}$ SNPs que mejor clasifican a 1043 individuos de un estudio público de diversidad humana conocido como HGDP [40] en categorías que representan grupos de etnicidades humanas consideradas estandarizadas en genética humana [41] en 3, 5 o 7 categorías.

Los métodos son: la implementación de MCTS descrita en 5.6 frente a otros dos métodos descritos en 5.7.

El primero de estos métodos es la utilización de los mejores clasificadores individuales según cada uno de los criterios descritos en 5.3 y que, individualmente, son los mejores métodos descritos en la literatura. En lugar de utilizar solamente el mejor, utilizamos el conjunto de los $n \in \{4, 8, 12\}$ mejores de cada criterio.

El segundo método es un método muy eficiente para encontrar los mejores modelos lineales de n factores. El algoritmo explora los diversos valores de n desde 1 hasta el valor deseado, realizando primero una rápida estimación del modelo en función de los resultados obtenidos por el modelo padre (el mismo modelo con un factor menos) y el factor que se añade al mismo. Esta estimación se convierte en la prioridad con la que el modelo va a ser finalmente evaluado. Los modelos van esperando en una cola y se evalúa a cada vez el de mayor prioridad. Además, si éste no tiene el número de factores deseado, sus hijos son añadidos a la cola. Normalmente, como en este caso, la regla de parada es un determinado tiempo. Con una capacidad de cola ilimitada y un tiempo ilimitado este algoritmo sería una búsqueda exhaustiva sin repetición.

2.4.5 Los resultados obtenidos

En todos los experimentos, los dos métodos basados en búsqueda superan en gran medida al método basado en la lista de los individualmente mejores. Dentro de los métodos basados en búsqueda, MCTS obtiene los mejores resultados en todos los problemas con $n > 4$ y también es el método que demuestra una mayor mejora con el tiempo añadido de CPU. Los resultados completos están descritos en 5.8.

Chapter 3. Applying knowledge in the tree search

This chapter reviews the most important ideas tested to produce successful MCTS implementations. The first two sections are not *go* specific; we describe four important ideas that can be considered in many game playing or optimization applications. In section 3.1 we introduce main ideas focusing on their motivation, and their influence on the algorithm, leaving the implementation specific details to the cited literature. In section 3.2, we describe mathematical models commonly used to deal with a priori information. After that, we introduce the most important limitation of MCTS as it is implemented today, both from a generic and also a *go* specific point of view. The rest of the chapter describes our original research in detail about using a full board evaluation function based on offline learned expert knowledge as a source of a priori information for the opening of 19x19 *go*. [1, 2, 42]

3.1 Successful ideas used in other programs

Below we describe some successful ideas that are used in many MCTS implementations, including: some techniques for **biasing** moves (i.e., combining a priori information with the results of the simulations), **RAVE** a technique that increases the amount of information available, **progressive widening** a technique that focuses the search in large problems initially to the a priori best candidates and we also discuss ways to implement **parallelization**.

3.1.1 Biasing moves

MCTS is a tree search and, just like in any tree search, the order in which the possible decisions (moves) are explored is essential, especially as the tree is wide and has many possible decisions that are not suitable. Finding heuristics that partially sort the set of legal moves is an obvious idea that has been present since the beginning of MCTS. This is sometimes named "progressive bias" as it was called in one of the early papers on the subject [25]. It is "progressive", as the authors of the paper intended, since in most cases, it fades out as the "real information" becomes less scarce. Quoting the original paper: *"The influence of this modification is important when a few games have been played, but decreases fast (when more games have been played) to ensure that the strategy converges to a selection strategy."* Some authors like David Fotland, the author of *The Many Faces of Go* have described their bias as permanent and it does not fade out at all: *"mfgo_bias is unchanging, per move, within a range of about ±2%, based on mfgo's move generator's estimate of the quality of the move."* [19]. Others authors just introduce some "fake" a priori simulations when the node is created and these "results" become naturally less important as the node is updated with real results. We use a separate term with an "a priori" estimate of the value of the node that plays a (decreasingly weighted) role until a horizon (in terms of number of real visits) is reached. After that, it does not interfere with the real information at all. The mathematical details of some described implementations are given in the next section.

In this thesis, we prefer the less specific "a priori information" to "progressive bias" to refer to the heuristics playing a role in determining the order of exploration in the nodes when simulations are scarce or nonexistent.

What heuristics are used as a priori information in the tree search is domain dependent, but in general, all heuristics used in the simulations like those described in 4.2, 4.6 and 4.7.5 are obvious candidates. The seminal papers include a number of simple heuristics [22, 25, 43] and also of more complex ideas including offline learned weights [24, 44]. Since then, other ideas have been described [45].

3.1.2 RAVE

Somewhat unexpectedly, since the precise order of the moves is vital in most domains (including *go*), moves also tend to be "generally good" or "generally bad" independently of the moment in which they are played. In absence of accurate information about how good a move is in a given situation, the knowledge about how good the move is overall is frequently better than no knowledge at all. The main advantage of this idea is

that the rate at which (even though inaccurate) information is updated is really high. Many moves are played in each simulation; hence, after a small number of simulations most moves will have information. This idea was already used in the pre-MCTS era with the name *All Moves As First* (AMAF) heuristic. In [46] the authors implemented the heuristic in GOBBLE a program that is the first documented Monte-Carlo *go* program dated back to 1993 [47]. Later "rediscovered" applied to tree search RAVE (Rapid Action Value Estimation) is now part of most strong MCTS engines. RAVE is the tree version of AMAF. The idea is updating the information kept in all sibling nodes of the played move (rather than updating only the played node) with the results of all moves played in the simulation. This, like in AMAF, increases the rate at which information is updated, but unlike in AMAF the information is specific for the board as it stands in the node from which the simulation was played. Different branches of the tree will (asymptotically) get different wins/visits proportions that are specific to the branch of the tree from which the simulations are played. But just like in AMAF, they are inaccurate since they do not represent the value of the moves played in correct sequences.

RAVE was described by its original authors in [21, 23]. Another study [48] analyzes different alternatives of information updating policies.

It is a heuristic used in most strong programs (all except Nomitan to our knowledge). It is usually weighted to fade out as the number of visits increases and combines very well with progressive widening (3.1.3) which is very important for large board sizes.

RAVE should not be considered *go* specific. In all domains in which the total number of legal decisions does not change completely for each game state, but is essentially a choice from a set (of manageable size) defined by all possible decisions, RAVE can be considered a valid heuristic. This is the case in the two optimization problems analyzed in this thesis: strip packing and the n factor GEP (Genetic Etiology Problem described in section 5.5). It is also the case of connection games like Hex, Havannah or TwixT and of many board games where moves are essentially board positions like Reversi, Amazons or Gomoku. This condition does not guarantee that RAVE will be useful, but at least, it does guarantee that RAVE will rapidly increase the rate at which by-move information is updated.

3.1.3 Progressive widening

As the width of the MCTS tree increases, the amount of nodes created without visits or with very few visits also increases. When this happens, if a priori information is weighted too heavily, we are losing the

advantages of tree search, i.e., having a model of answers that contains game outcome information from "likely" follow ups. Results are biased by preconceptions that may or may not be applicable to the position analyzed. If, on the other side, we weight a priori information too weakly, the combinatorial explosion of meaningless nodes would make the algorithm need a huge amount of time and memory until it starts following meaningful sequences. A solution to this problem is limiting the width considered to a list of few (3 to 6) moves and not widening this list until the parent node has received enough visits. This idea has an additional advantage when combined with RAVE: RAVE updates parent board specific information not just for the moves played, but for all their siblings since they will frequently be played later in the playout. When the width is widened, the candidates to be the next move explored do not only have a priori information but they also have position specific RAVE information. When this information is used (in combination with a priori information) to decide which move is the next to be analyzed, RAVE can pick up a move that was not near the top according to a priori ranking but is showing very successful (although ignoring precise move ordering) in the playouts. This, like RAVE itself, helps finding good candidates.

Progressive widening was first described in [24]. The original description is somewhat different from the one given here. Rémi Coulom has a specific tree expansion policy in which he is gathering information before actually adding the nodes to the tree. When the nodes are "promoted to internal node" the heuristic works just as described above:

"When a node in the Monte-Carlo search tree is created, it is searched for a while without any pruning, selecting the move according the policy of random simulations. As soon as a number of simulations is equal to the number of points of the board, this node is promoted to internal node, and pruning is applied. Pruning consists in restricting the search to the n best moves according to patterns, with n growing like the logarithm of the number of random simulations. More precisely, the n-th (n ≥ 2) move is added when $40 \times 1.4^{n-2}$ *simulations have been run. On 19x19, because of the strength of the distance-to-the-previous-move feature, progressive widening tends to produce a local search, again like in Mogo."*

Using the logarithm of the number of parent visits to determine the width, as the original author does, has become common practice. Most implementations use equation 3.4 with some tuned constants:

$$N_{pw} = K_s \cdot \log(K_f . n_j) + N_{min} \tag{3.4}$$

N_{pw} is the number of children of the parent node j to be explored and n_j is the number of visits of j.

Hiroshi Yamashita, author of the program Aya, reported $K_s = 1/\log(1.4)$, $K_f = 1/40$ and $N_{min} = 2$ as the values used by Aya. Our program has a configurable categorical factor setting the values to narrower/wider exploration in five categorical steps, the intermediate setting corresponding to Aya's values with $N_{min} = 4$.

3.1.4 Parallelization of MCTS

MCTS has very suitable qualities, one is being an *anytime* algorithm. Anytime algorithms have naturally good parallelization. In this case, parallelization consists in running the same algorithm in different CPU-cores or computers and sharing information.

Parallelization algorithms fit in two categories:

- Shared memory. One multi-core computer runs a multithreaded application.
- In a cluster. A cluster of computers sharing a fast network use an MPI (Message Passing Interface) to keep nodes in synch.

We did not implement cluster parallelization in our program. Some open source programs a have cluster parallelized version including Fuego [49] and Pachi [50]. Hideki Kato, who successfully implemented cluster parallelization for the program Zen, was one of the early researchers on cluster parallelization [51]. The ideas implemented range from root parallelization (the term is used for either strictly the root nodes or the set of k nodes with the highest number of visits) where the results of a small set of nodes are synchronized via MPI messages, to more advanced algorithms [52]. Some implementations use more ambitious paradigms like a single tree shared among many computers, in which the computer owning the leaf node runs the simulation. This has been successfully implemented by Lars Schäfers, author of Gomorra and is still unpublished by 2012. Studies on root parallelization applied to *go* include [53]. A paper describing both shared memory and cluster MCTS parallelization methods was published recently by a *go* program author [54].

As far as shared memory parallelization is concerned, one of the early works [55] describes various methods. The study is conditioned by considering thread locking as a serious limitation. This has been overcome by implementing lockless schemes. The work is also worth mentioning as the ideas described in it have influenced cluster parallelization. At the beginning of 2009 we were already working on lockless

parallelization which allows the algorithm to run without any measurable interlocking and with the probability of results being lost kept so low that its impact cannot be measured experimentally. The key idea is using CPU instructions reading/writing variables stored in the nodes always atomically. Also, incrementing the number of visits by the use of the *lock inc* CPU instruction. Note that this is instruction level locking only, it makes the whole read/update/write cycle atomic. It should not be confused with thread locking in which a thread waits for some condition. Our program is one of the first three implementing lockless tree update. We had already implemented it when the idea was mentioned for the first time by Rémi Coulom in [19]. That same year the Fuego team described it a conference whose proceeding were published the following year [56]. Today, all strong engines implement a lockless shared tree when running on a single computer. In terms of playing strength, lockless tree sharing scales (by identical number of simulations) just like playing the same number of simulations in sequence. In terms of real time CPU usage, the improvement is less than expected from the number of threads, since memory and CPU cache sharing produce a global slowdown resulting in less simulations being computed than a linear model would predict.

It is also worth noting that shared tree parallelization is not independent from the exploration/exploitation equilibrium. Many threads simultaneously exploring the same tree favor exploitation since in the beginning the deterministic part of the UCT algorithm will chose the same path. This can be compensated either by setting the UCT constant to a value including more exploration or by inserting a virtual visit as the tree is travelled downwards. An additional visit (without a win) acts like a virtual loss and favors exploring alternative paths in the tree. When the simulation is done, either a win or a loss is updated (without adding another visit, obviously) keeping the information to date. This idea, known as virtual loss, was also described in [55].

3.2 Including a priori information in the tree

The description of the MCTS algorithm includes selection at each step the action a^* maximizing a value:

$$a^* = \underset{a}{argmax}\, V(s,a) \tag{3.5}$$

s being the state of the game at the node, (i.e., the board position, the player in turn and the moves made illegal by repetitions, if any). In this section we analyze different functions used to define $V(s,a)$. This procedure is used to select the next node to be visited while exploring the tree, but it is not a good idea to use it to determine what the best move at root is since it may contain an "exploratory" node whose value is not well established. When the search has been completed, the "standard" policy is selecting the move with the highest number of visits which is the best decision "so far".

3.2.1 Mathematical models described to deal with a priori information

Below, some models are described ranging from the simplest model without RAVE to more complete models including separate a priori information terms.

The first model which does not include RAVE or a priori information is the "vanilla" UCT formula:

$$V(s,a) = \frac{w(s,a)}{n(s,a)} + K \cdot \sqrt{\frac{\log(n(s))}{n(s,a)}} \tag{3.6}$$

where $n(s,a)$ is the number of visits stored in each child node for action a at state s, $n(s)$ is the number of visits in the parent node representing state s, $w(s,a)$ is the number of wins for action a at state s and K is a constant defining the exploration/exploitation tradeoff. This model was proposed in [20].

The first introduction of RAVE was proposed in [21] and we are not presenting its mathematical details here as the function was improved by its original authors, first advanced in [57] and finally published in a revision article on RAVE in [23]. The improved version, which we include below, analyzes the variance and the bias introduced by RAVE. Like in the analysis detailed below in 4.1.1, the simulation is the repetition of a Bernoulli process; hence its variance is determined by the binomial distribution. This analysis does not study the underlying Wiener process and treats the simulation as unbiased without further questioning, considering the value to which the simulations asymptotically converge the "true" value of the position and, since it does not match the value obtained from RAVE, the latter is considered biased in the sense: $E[Q_{UCT} - Q_{RAVE}] \neq 0$. This bias is given the name b_r reestablishing: $E[Q_{UCT} - Q_{RAVE} + b_r] = 0$. Q_{UCT} is the value obtained from UCT (without any confidence interval), i.e., the proportion $wins/visits$ for the move and Q_{RAVE} is the value estimated using RAVE.

Also, the author mentions: *"The RAVE estimate has a lower variance, but a higher bias than the UCT estimate. When there is little experience, the RAVE estimate is more reliable; when there is more experience, the UCT estimate is more reliable."* It is worth noting that the smaller variance of RAVE is only expected because of its higher number of visits, being both variances following the binomial distribution:

$$\sigma^2_{UCT} = q_u \cdot (1 - q_u)/n \tag{3.7}$$

$$\sigma^2_{RAVE} = q_u \cdot (1 - q_u)/m \tag{3.8}$$

Where $n < m$ are the number of visits of the UCT evaluation and the RAVE evaluation respectively, and q_u is the "true" value of the node from which the proportions of RAVE wins is a biased estimator, hence the same in both equations, simplifying the analysis.

The Gelly and Silver suggest a value $V(s,a)$ combining both proportions in a convex sum:

$$V(s,a) = \beta \cdot V_{RAVE}(s,a) + (1-\beta) \cdot V_{UCT}(s,a) \tag{3.9}$$

being:

$$V_{RAVE}(s,a) = \frac{w_{RAVE}(s,a)}{m(s,a)} + K_r \cdot \sqrt{\frac{\log(m(s))}{m(s,a)}} \tag{3.10}$$

$$V_{UCT}(s,a) = \frac{w(s,a)}{n(s,a)} + K_u \cdot \sqrt{\frac{\log(n(s))}{n(s,a)}} \tag{3.11}$$

analogous definitions to equation 3.6. $m(s,a)$ is the number of RAVE visits for action a at state s, $w_{RAVE}(s,a)$ is the number of RAVE wins for the same state, $m(s)$is the sum of the number of RAVE visits of all siblings (stored in the parent, but it is not the same as the number of RAVE visits of the parent), K_r and K_u are two constants.

The whole point is finding the value β that minimizes the MSE (mean squared error) in equation 3.9.

$$MSE = \sigma_{ur}^2 + b_{ur}^2 = \beta^2 \cdot \sigma_r^2 + (1-\beta)^2 \cdot \sigma_u^2 + \beta^2 \cdot b_r^2 \tag{3.12}$$

Differentiating with respect to β, equaling to 0 and isolating β:

$$\beta = \frac{\sigma_u^2}{\sigma_u^2 + \sigma_r^2 + b_r^2} = \frac{m}{m+n+m \cdot n \cdot b_r^2 / q_u \cdot (1-q_u)} \tag{3.13}$$

The whole $b_r^2/q_u \cdot (1-q_u)$ term is just a constant representing a hypothesis about the bias b_r^2 and the true value of the node q_u. In Pachi [50], the value is set to $b_r^2/q_u \cdot (1-q_u) = 1/3000$. Yoji Ojima, known as Yamato, the author of the *go* program Zen [58], reported $b_r^2/q_u \cdot (1-q_u) = 1/400$ for Zen.

Hiroshi Yamashita, the author of the *go* program Aya [59], reported using a simplified version of β with $K_r = K_u = 0.31$

$$\beta = \sqrt{\frac{100}{3 \cdot n + 100}} \tag{3.14}$$

without using m at all.

This mathematical framework is used in many strong programs. Since it does not provide an independent term for a priori information, it is usually introduced by adding some virtual wins and visits to the node when creating it rather than initializing it with 0 wins, 0 visits. Other programs like, as mentioned in 3.1.1, Many Faces Of *Go* [60] use a specific bias term added to this value.

3.2.2 A priori information and tree search as implemented in GoKnot

Now, we describe the mathematical model as implemented in our own program, GoKnot (described below in 6.1). This is our original research. The framework just described, although mathematically sound, is based on some oversimplifications and has the disadvantage of not allowing distinguishing between a priori information and information really obtained from the simulations. We wanted to keep these two values separate for two reasons:

- We hypothesize that bias introduced by virtual a results has negative impact when the a priori hypothesis assumed by the heuristic has been proven wrong by the simulations.
- We want to be able to judge the quality of our a priori information to tune it. For this reason we use some metric like Spearman rank correlation to measure "goodness of fit" between a priori values and values obtained from simulations.

The main equation in our framework is:

$$V(s,a) = \alpha \cdot V_{ap}(s,a) + \beta \cdot V_{RAVE}(s,a) + (1-\alpha-\beta) \cdot V_{UCT}(s,a) \qquad (3.15)$$

Where $V_{ap}(s,a)$ is the a priori value assigned to the node by some heuristics when the node is created. These heuristics, tuned using CLOP [61], give weights to features described in chapter 4 such as: *atari*, extend from *atari*, *self-atari* (with negative bias), urgent answer patterns, eye improvement heuristics and also the *joseki* patterns described below in 3.4.4. $V_{RAVE}(s,a)$ does not include a confidence interval:

$$V_{RAVE}(s,a) = \frac{w_{RAVE}(s,a)}{m(s,a)} \qquad (3.16)$$

β is as in equation 3.13 , $V_{UCT}(s,a)$ is as in equation 3.11 and

$$\alpha = \max(0, \frac{n_H(s,a) - n(s,a)}{n_H(s,a)}) \qquad (3.17)$$

being $n_H(s,a)$ the horizon for the a priori information term. The weight of the a priori information decreases linearly until the horizon is reached and that decreasing weight is increased to the empirically determined value $V_{UCT}(s,a)$ which in the end becomes the dominant term.

3.3 Current limits of MCTS

This section is a description of the current limits of MCTS. It is included here rather that in the final discussion because it is the motivation of the two main contributions of this PhD dissertation to computer *go* described in detail in 3.4 and 4.5 to 4.8.

3.3.1 Limits in terms of computer science

On the good side, MCTS has been described [27] as: **aheuristic** (it is amazing how a prototype vanilla MCTS implementation outperforms state of the art, domain specific implementations in problems like strip packing), **asymmetric** (tree growth that adapts to the topology of the search space), **anytime** and **elegant**. We wish to add **analytically sound**. Unlike some randomized methods (including GA and swarms) in which mathematical analysis is not usual, MCTS can be analyzed by equations, as we do all through this dissertation. MCTS combines the "sense of winning" of Monte-Carlo methods found out naturally without following any rules other than the rules of the definition of the problem with the "sense of answering" of tree search.

This way to look at the whole picture was pointed out by Yamato, the main author of the *go* program Zen, the strongest nowadays. He used the Japanese words *yomi* which means reading for the tree search and *kankaku* which is feeling for the evaluation from the playouts. Note that a *go* engine is not doing something so different of what human do. It read sequences of plausible moves and when it reaches the end of the tree, it does a random experiment which represents the *kankaku* i.e., the feeling it has about the resulting position. Of course, humans do it few times and very accurately and computers do it thousands of times and only the whole picture makes sense.

But even if MCTS represents a breakthrough in many hard problems (the list of domains increases every day as described in 7.1) and finds out its answers unraveling the problem naturally without explicit rules, two limitations still apply:

- MCTS being an online learning method, there is no simple way to propagate the information learned in the tree either across the tree itself (the same local problem has to be solved many times) or to the simulations which are usually too simplistic to get evaluation right until the position is "almost settled".

- MCTS cannot handle long sequences accurately. Even when the most expected answer is correct 80% of the times, the probability that a sequence of 10 consecutive correct answers is followed is around 10% of the times. When the correct sequence is required to get the correct answer, the wrong answer is obtained 90% of the times making the whole idea unworkable.

The first limitation is the reason why MCTS programs weaken when the number of unsolved tactical problems increases on the board. MCTS is strong enough to solve each problem in isolation, but as the search jumps from one problem to another the problems have to be rediscovered too many times wasting large amounts of resources. Ways to (hopefully) work around this problem range from divide and conquer ideas mentioned briefly in 3.3.4 to learning playouts. Our original work in learning playouts is described in detail in the next chapter.

The second (made worse by the first) explains most of the limits in terms of *go*. It cannot be directly avoided by, for instance, making the tree search less exploratory. The tree search algorithm needs to explore alternatives. How else could it know that a move is the best answer if it did not explore the alternative moves to it? For that reason, even in the best known part of the tree, the path does not include always the correct answer. This is inevitable, note that the whole exploration/exploitation paradigm has been researched deeply. Simple ideas that were acceptable few years ago like ε-greedy search are ridiculously inefficient when compared to UCT. The problem is even worse since, due to the scarce representation of the "correct answering lines" in the entire search, they do not guide the growth of the tree in "correct directions" at all, neither is there a way to determine what paths are correct. The amount of search required to establish a ten move long path of not obvious moves requiring a precise order is so large that it may be impractical in most cases to "just wait" until MCTS finds the path.

3.3.2 Limits in *go* terms

In terms of *go*, this difficulty with long sequences explains what players see:

- Problems with *semeai*. The evaluation of a *semeai* will only be correct when it is played in reasonable sequence. The correct sequence is usually not unique, but it is still unlikely to find the sequence in the space of all legal sequences.
- Problems with *ko*. The value is only correct when the entire *ko*/threat/response/recapture/etc. sequence is played in meaningful orders. In plural, since different paths have to be explored to get a representation of the value of everything and the interactions. MCTS engines can make things even worse by wasting threats which appear unnecessary but playable moves when they are not playable at all. When placed in correct sequence, they decide the outcome of the *ko*.
- Close *endgames*. The engine will not get the correct evaluation since these endgames depend of sente and gote. Playing the whole sequence in one order may differ from playing it in another order by a few points and, when these points decide the game, the engine does not see the winning path frequently enough.

3.3.3 The opening as a limitation

And here comes the important point justifying the need of opening knowledge: The opening is just another case in which the MCTS engine is weak and for the same reason as in all the others cases. The engine does not follow long sequences accurately. In other words, it cannot evaluate territory in the sides and the corners as humans who know *joseki* and opening principles can.

Strong human players who "specialize" in beating the strongest engines (although their playing strength is about the same as that of the engine) have declared in public [62] that when they don't get advantage in the opening, they just resign because later the engine becomes just too strong.

Note that the length of a played out game from the opening is, in our implementation, between 501 and 552 moves long 50% of the time as shown by its IQR (interquartile range). The tree search is only exceptionally like 20 moves deep. In the opening, it is much less because few moves are forced. This means the tree is just floating in over 500 moves of random *kankaku* which is *kyu* level even in the strongest engines. How could the tree search understand the position without additional help?

MCTS has a good sense of local tactics and it can be biased to explore *joseki* sequences coded as patterns. This may make its play look strong to some players, but that still does not result in correct evaluation of neither territory nor influence. And this lack of whole board understanding is never forgiven by strong players against which the strongest engines do not have the slightest chance in an even game. Even if MCTS evolved to be stronger in the middle and end game than humans, it would still have to recover from a disadvantageous position because it has ignored opening principles. This forces it to take higher risks later.

It was controversial when we started with this ambitious implementation in 2009 while our engine was still weak, and M-eval was certainly not the "lowest hanging fruit" to improve it, if the idea was necessary at all. At the time, many critics considered that opening played by strong MCTS engines had no room for improvement, probably carried away by the fast early success, without giving the matter a deep insight.

Nowadays, the opening is still a weak point in strong MCTS engines and, as our engine has evolved, the whole idea has evolved with it. It is still an improvement in both style and performance. It is now driven by *joseki* patterns and it could finally become a correct way to implement *joseki* patterns in 19x19 *go*, rather than simply biasing moves that may or may not be applicable considering the board as a whole.

3.3.4 Divide and conquer

Most of the problems resulting from the first limitation described in 3.3.1, i.e., the tree not benefiting from the knowledge acquired in other parts of the tree related to the same local sequences, would be bearable if an efficient board division algorithm was known. Although important results were obtained in the pre-MCTS era, it must be noted that they were applied in the late endgame or in artificially created problems were isolating local fights is feasible. Unfortunately, a realistic *go* engine requires the division to be understood much earlier.

Some approaches are worth mentioning, with success limited to solving specific problems like tsumego and endgame. To our knowledge, no strong program uses separate local searches at the moment since combining evaluation and not misrepresenting the interactions between local fights is extremely hard. In the case of non-interacting local fights, CGT (Combinatorial Game Theory) a very elegant mathematical theory of *go* introduced first by Elwyn Berlekamp in [63] and extended in [64] is worth mentioning. It was influential in many "classical" computer *go* work including the PhD dissertation of Martin Müller [65] a computer *go* pioneer, co-author of Fuego [49].

To define the problem of what is a local fight some words on board metrics may be necessary. Usually, humans use the word "local" to define a set of chains representing a subproblem that can be analyzed independently, although it usually interacts with other subproblems by interactions (sente/gote, possible *ko* threats, etc.) that can also be analyzed. Defined mathematically, the idea is much more pedestrian and usually refers to the region on the board whose distance to some point is below some value according to one of the following metrics for the distance between two points (x_1, y_1) and (x_2, y_2) (from worst to best) :

- Manhattan distance: $d_M = |x_1 - x_2| + |y_1 - y_2|$ An obvious choice in discrete spaces.
- *Go* board distance: $d_G = |x_1 - x_2| + |y_1 - y_2| + \max(|x_1 - x_2|, |y_1 - y_2|)$ This metric has been used in many computer *go* applications since at least 20 years. We were unable to identify its original author hence naming it *go* board distance. It works better than the Manhattan distance in representing the idea of influence.
- Common Fate Graph Distance [66] is a step in the correct direction to represent chains on the board. As all stones have a common fate they are linked counting the distance to the nearest stone in the chain. Usually a table of distances from each intersection to each intersection is built, rather than computing the distance each time it is needed. It is built by a filling algorithm. The method starts from one string and recursively adds close empty intersections and strings close to these empty intersections until no more close strings are found within a distance controlled by a parameter. Strictly speaking, it is a pseudometric since $d(A, B) = 0 \nRightarrow A \equiv B$ as the distance between two intersections of the same chain is zero. Figure **3.1** shows an example of CFG distance. All intersections at a distance ≤ 4 from A, display their distance to A.

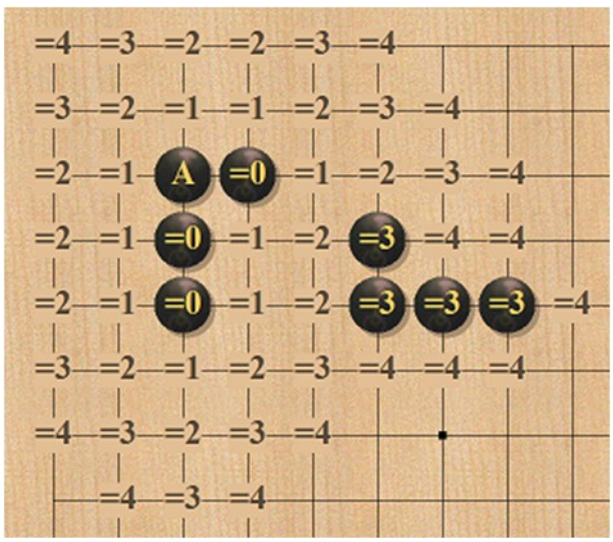

Figure 3.1. Intersections at CFG distance ≤4 of point A.

3.4 M-eval: A Multivariate full board evaluation function

This section describes our original work for implementing an evaluation function for opening positions named M-eval described in [1] and later in [2]. In the whole section the board size is 19x19.

3.4.1 Positional judgment in *go*

Evaluating a board position in *go* is extremely difficult even for professional players. All evaluation functions used in computer *go* have limitations and produce erroneous results under certain circumstances.

Most of the strength of MCTS comes from the fact that it does not require an explicit evaluation function. The Monte-Carlo evaluation is the result of a stochastic process, i.e., the complete playout of a random game. Assuming the program understands particular circumstances like *seki* (see 4.3.3), the evaluation of the final position is error free. As the end of the game approaches, given enough simulations, the tree reaches final or near final positions. *Yose* played by strong MCTS engines is equivalent to that of strong players as long as the outcome of the game is undecided and the correct move sequences are not too long to be followed frequently. In the middle game, MCTS engines show strength due to full board evaluation and often overcome losing local fights making excellent use of *tenuki* and *furikawari*.

Despite their strength, MCTS programs cannot evaluate opening positions accurately as explained in 3.3.3. The length of the playout games is (mean = 528, interquartile range (IQR) (501—552)) moves to reach a final position (i.e., without remaining moves that are not restricted by a rule). This does not allow the tree to reach near endgame positions since the root position contains few moves (e.g., 30) and the maximum length in the tree is at most 10 to 15 moves as there are many legal moves and few obvious high priority sequences such as killing a big group. Since the whole evaluation is based on self play, MCTS programs are essentially blind to deviation from opening principles as both sides do the same.

In game programming, an opening book is a database of full board positions linked to a list of playable moves for each position. Opening books can be either created by experts or mined from large sets of high quality games extracting the moves seen many times. Full board opening books in 19x19 *go* have been used by many authors including [49, 67, 68]. We have also implemented a book for time saving reasons in tournament

games. Only exceptionally more than eight moves (four per side) are played from the book. This represents a very small part of the opening. Partial (*joseki*) pattern matchers have much longer application, but deciding which *joseki* should be applied is a very complicated problem where bad choices can be really bad.

Many books describe *go* opening principles [69-72]. Cho Chikun [10] describes a method for counting territory, estimate *thickness* and *moyo*. Also, other researchers have developed positional judgment functions [73, 74]. The strength of a *go* program depends on entire sequences, not just in finding good moves frequently. Therefore, algorithms with good professional move prediction rates can produce weaker programs than simple but consistent ideas (see 6.1.3) like those described in 4.4. It is hard to tell which of the resulting moves are going to be "understood" and successfully exploited by MCTS, our intention is providing usable a priori information to MCTS that increases its strength in the opening.

3.4.2 M-eval rationale

To explain the motivation underlying our algorithm, the following aims must be kept in mind:

a. *Go* is a game of balance, e.g., balance between speed and connectivity, between territory and influence, etc. Too much connectivity (known as slow play) gives the opponent the opportunity to consolidate a territorial advantage, too much speed (known as overplay) leaves exploitable weakness behind. Strong play in *go* is about achieving goals in many different directions always considering the achievements of the opponent. There is a natural equivalence between obtaining a gain $+g$ and destroying an equivalent gain $-g$ from the opponent's achievements. Sometimes the former is easier, sometimes the latter. Positional judgment is multivariate in nature.
b. The relation between the move and the next moves is fundamental. Sometimes the best reason for a move is not letting the opponent play that same move. To take the interaction with other moves into consideration, positional evaluation requires search. Search finds strategic *go* principles like *miai* naturally. When there is an alternative way to reach the same target, urgency decreases automatically. Without search, such concepts are hard to define and subject to misevaluation.
c. Perfect positional judgment (minimax wise) is not intended. The positional judgment evaluation proposed here is designed to interact smoothly with MCTS. Rather than determining what the "best" move is, it aims to guide the search towards directions that are compatible with human *go* opening principles and consistent.

d. Any mathematically definable evaluation function implemented in a program will be far from flawless. Accepting the limitations of our individual functions, we need more than one way to measure each abstract strategic concept. Because of the excess in degrees of freedom, dimensionality reduction is mandatory. Furthermore, the first step in this reduction has to be based on the knowledge contained in a massive dataset.
e. Since, without any loss of generality, functions can be defined as positive for each player, named *qualities* (e.g. enclosed territory, influence, thickness, more liberties in a *semeai* race, etc.), it would be very hard to justify that a dimensionality reduction technique translates a positive quality by multiplying it times a negative coefficient. That may reduce the squared residuals or some other metric, but it does not make sense as an evaluation function. In short, we have to preserve the non-negative nature of our *qualities*.

3.4.3 NMF: Non-negative matrix factorization

Originally introduced by Paatero et al. [75, 76] NMF has extended its use to many fields in data analysis, including image analysis. The property of NMF to decompose objects into parts was described by Lee and Seung [77] and further studied by Donoho et al [78].

Given n board positions for which p qualities have been measured for each player, these non-negative values can be written as a matrix $X_{n\times p}$ which can be decomposed as:

$$X_{n\times p} = W_{n\times d} \times H_{d\times p} + U_{n\times p} \qquad (3.18)$$
$$\text{Subject to: } W_{n\times d} \geq 0, H_{d\times p} \geq 0$$

When the residue $U_{n\times p}$ is minimized, $V_{n\times p} = W_{n\times d} \times H_{d\times p}$ is an approximation of $X_{n\times p}$ called the NMF of $X_{n\times p}$ and $d < p$ is the dimension to which the dataset $X_{n\times p}$ is reduced.

$W_{n\times d}$ is a matrix whose rows (named *encodings*) are compressed images of the rows of $X_{n\times p}$.

Since our intention is using $V_{n\times p}$ to represent the entire dataset, we want to minimize the informational loss resulting of using $V_{n\times p}$ instead of $X_{n\times p}$. That loss is proportional to (it would be equal if $\sum_{i,j} x_{ij} = \sum_{i,j} v_{ij} = 1$) the Kullback-Leibler [79] divergence $D_{KL}(X|V)$, i.e., the extra message-length per datum that

must be communicated if a code that is optimal for a wrong distribution V is used instead of a code based on the true distribution X.

Precisely: Rather than minimizing D_{SE}, the squared error of the residue $U_{n\times p}$, $D_{SE} = \sum_{i,j} u_{ij}{}^2$ we minimized $D_{KL} = \sum_{i,j} \log\left(\frac{x_{ij}}{v_{ij}}\right) x_{ij}$ which is the same expression as the Kullback-Leibler divergence, but without the additional condition requiring X and V to be probability distributions. This is common practice in NMF and included as standard in the Matlab library we used [80].

We computed the NMF of our dataset using the Lee and Seung algorithm [81] version that uses $D_{KL}(X|V)$ as the cost function. Furthermore, $H^{+}_{p\times d}$, a pseudo-inverse of $H_{d\times p}$, was computed for the online conversion of any point $X_{1\times p}$ from the initial $\mathbb{R}^p$ space to its encoding $W_{1\times d}$ in $\mathbb{R}^d$, using:

$$W_{1\times d} = X_{1\times p} \times H^{+}_{p\times d} \tag{3.19}$$

Finally, the $H^{+}_{p\times d}$ matrix is scaled to standardize the encoding over the dataset of 110,542 cases created taking 2 positions from each game between moves 16 and 62 from the database 6.1.2. I.e., when $H^{+}_{p\times d}$ is multiplied times the entire dataset measured for the differences between both players, it returns a dataset of d variables and 110,542 cases, each variable having $mean \approx 0$ and $SD = 1$.

3.4.4 Qualities implemented in M-eval

We have experimented implementing M-eval in different versions. The first successful implementation was in 2009 on our first MCTS *go* program QYZ, now called isGO [1]. In two years the program strength has increased by tuning, bug fixing and the implementation of other heuristics, primarily RAVE [21, 48] and progressive widening [24, 25]. isGO is now a state of the art MCTS engine. The current level in 2012 is around 100 Elo points stronger than the version achieving the 4th place in the slow KGS 19x19 computer *go* tournament in 2011. We describe the first implementation on which we performed experiments as version 1 and the last one as version 2 although evolution has been continuous.

Following aims a, d and e in 3.4.2, board positions are evaluated by computing a set of p qualities $X_{1\times p}$ for each player. That vector is dimensionally reduced to d components by equation (3.19). Then, $W_{1\times d}$ is reduced to a real value v by:

$$v = W_{1\times d} \times L'_{d\times 1} \tag{3.20}$$

Where $L'_{d\times 1}$ is a direction learned online by a version dependent algorithm (see 3.4.7).

The value of the position is the difference in v measured for both players.

Terms used by human players, such as: strength, potential territory, influence, connectivity, etc. are abstract and do not have a mathematical definition. We define *qualities* which are mathematical functions both users want to maximize. Since these qualities are different attempts to measure the abstract underlying variables, they are highly correlated. We did parameter tuning analyzing the distribution of each proposed quality using a dataset of high level play positions. We also eliminated candidate qualities analyzing their redundancy and their use as regressors on a dataset of intentionally unbalanced positions to measure their capacity to predict positional advantage. This work was mainly done by trial and error trying to combine simple definitions with the maximum available information, including the information available from urgency patterns and from *ownership maps* (i.e., the proportions of black/white ownership of each board intersection learned from previous simulations).

Version 1 (the M-Eval version implemented in 2009 and published in 2010 [1]) implemented two types of qualities: *deterministic qualities*, which are only a function of the board position and *stochastic qualities* which combined the board position with previous knowledge of the ownership of each board cell. This ownership was computed from previous simulations. This idea intended to have a measure of the strength of stones. The drawback was it complicated implementation by requiring updated ownership information. A series of experiments, as the strength of the program improved, showed that removing stochastic qualities produced no measurable loss of strength. We conjecture that this happened because correlated qualities existed in the deterministic set. Another important modification was the implementation of two new qualities that estimate territory around the corners and the sides. This is done by regression of stone patterns into final positions of master games evaluated by isGO. Results from this estimation are shown in 3.4.9. This quality replaced previous territory measures used in version 1. The full list of 19 qualities of version 1 are described in [1]. Version 2 (the M-Eval version developed until 2011 and published in 2012 [2]) implements the 8 qualities described in table 3.2.

Table 3.2. Qualities used in version 2 M-eval

Name	Related *go* concept	Description
terriCorner	Territory	Estimated territory in the corner for the current player minus that of his opponent. The estimation uses 20,201 big corner and 46,979 small corner patterns seen more than 5 times in the master's database. To each pattern found the appropriate territory is assigned subtracting the standard error of its mean. That territory was computed from the final positions of the games in the database by isGO.
terriInSide	Territory	Estimated territory in the sides for the current player minus that of his opponent. The estimation uses 27,881 big side patterns seen more than 5 times in the master's database computed as in terriCorner.
strenStones	captures, strength	Number of stones for the current player minus those of his opponent.
strenLibert	Strength	Number of liberties for the current player minus those of his opponent.
strenAdjLib	*Semeai*	Compensated difference (own - opponent) in liberties for each adjacency. An adjacency is a set of 2 groups of different color with stones in contact. All different adjacencies are considered.
strenUrgent	shape, strength	Weighted sum of urgent points for each player. Bad (= urgent) shape is not about having very urgent cells, but about having many. Urgency is read from a database of 40 cell patterns learned from *dan*/pro level play.
speedRows34	Influence	Number of empty cells in rows 3 and 4 within the reach (Manhattan distance ≤ 4) of own stones.
speedHigher	Influence	Number of empty cells above row 4 within the reach (Manhattan distance ≤ 4) of own stones.

3.4.5 M-eval search

Following aim b in 3.4.2, the M-eval value used for each candidate move m_i is the evaluation resulting of a short search starting from that position, rather than the evaluation of just the position after playing m_i. This measures the importance of moves including considerations such as *miai* that are not coded in the evaluation function. In version 1 we hypothesized that blurring deterministic evaluation errors by inserting a brief stochastic component (a random sample of positions following the root position) was important. The original search was interrupted after a small number of moves (4 and 8 moves were tried) and the final node evaluated as in equation 3.20. We compared this idea with a short minimax search and the latter proved to be superior

both in tree metrics and in self play competitions. The minimax search is 4 nodes deep at root but applies only to local answers generated by the *joseki* patterns and is therefore fast enough. Deeper in the tree, the search depth is shortened by each additional depth level and, below tree depth 4, M-eval is not used.

Following aim c in 3.4.2, the result of M-eval is not used directly to choose the move played, but integrated with MCTS search. In version 2, when the main MCTS algorithm expands a node, the M-eval object is called only if it returned valid information for the parent of the node being expanded. If the M-eval object finds known patterns for the corners and sides of the current position, it will search the moves corresponding to $(state) \rightarrow action$ pairs stored for those patterns in the database. The leaves of this short M-eval search are evaluated by the M-eval function as in equation 3.20. The resulting value is converted into virtual wins and losses that seed the newly created tree node by subtracting average evaluation and seeding only the positive values. This conversion uses configurable parameters for *persistence* (number of virtual visits) and *intensity* (linear scale translating values into wins / losses ratio).

M-eval no longer needs being disconnected after the opening. It automatically evaluates just known patterns and local moves suggested by patterns (which support multiple actions). After no known patterns remain on the board, it will stop generating a priori information and no longer be queried for the descendants of nodes for which it did not provide information. It can gracefully handle complicated boards where some parts cannot be evaluated and other parts have simple shapes. The M-eval object will only provide information for patterns it can understand. Even if M-eval is a full board evaluation, when boards include not well understood areas, these areas produce constant evaluation as no moves will be generated in them during search. Since average evaluation is subtracted, the search is guided by local achievements. Currently, M-eval does not try to find out absolute values of the positions, instead it finds out relative values of moves frequently played by humans around known patterns. These values seed the MCTS search "most promising first".

3.4.6 Offline learning of $H^{+}_{p \times d}$

For using dimensionality reduction techniques, it is necessary that the complete dataset can be approximated with fewer variables. Since board positions of the same game are not independent samples, we decided to take only 2 positions from each game between moves 16 and 62 from a database of 55,271 games played by *dan*-level and professional *go* players (see 6.1.2). This dataset of 110,542 board positions is evaluated for the current player (positive qualities) and also subtracting the values measured for his opponent.

A Principal Component Analysis (PCA) shows that both versions are good candidates for dimensionality reduction techniques.

As shown in figure **3.2**, the first four principal components in version 1 explain 75.8% of the total variance, and the first two in version 2 explain 42.5%. Note that these numbers cannot be compared directly. The former qualities have more variables aiming at the same measure in two ways (deterministic and with board ownership information). The information produced by the latter qualities is less redundant and more reliable.

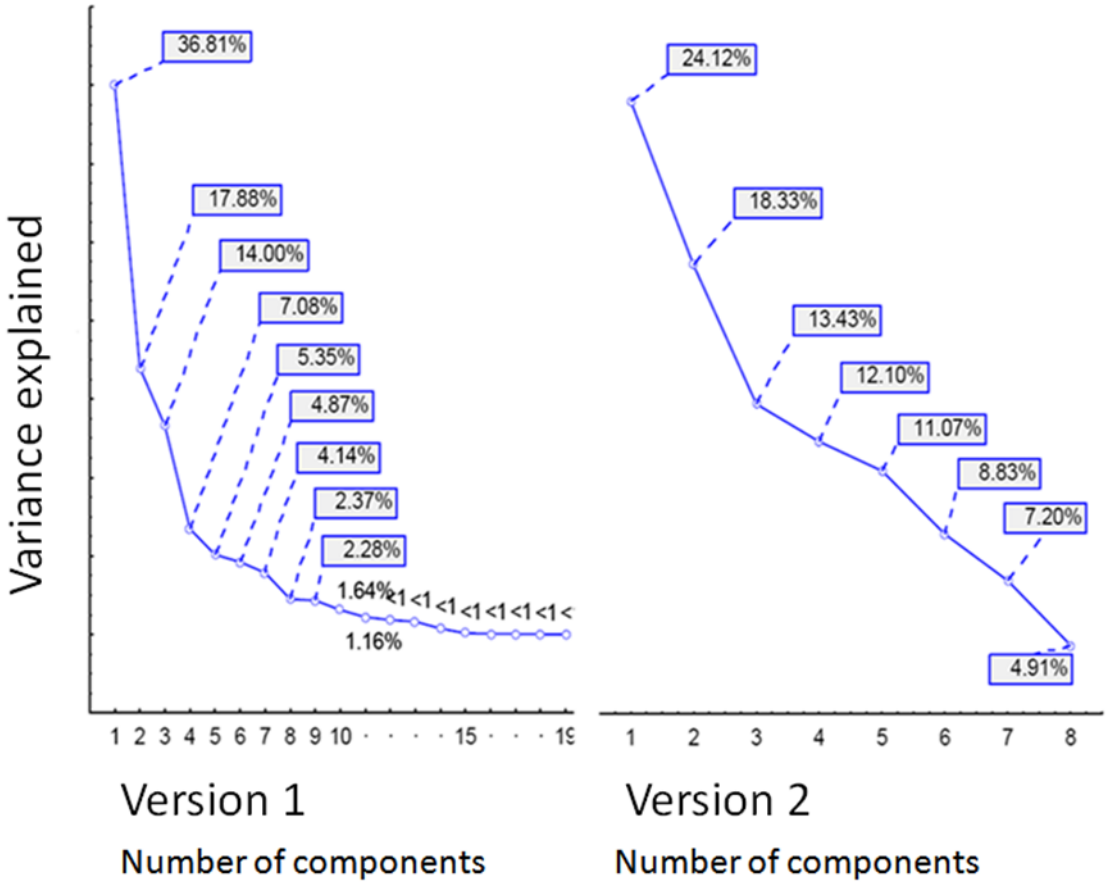

Figure 3.2. Principal Component (PC) analysis of the *qualities* over the learning dataset.

We used PCA only to analyze the multivariate structure in order to decide the rank of the encodings matrix, not for reducing dimensionality. Our first choice was $d = 4$ preserving the maximum amount of information. This produced too many degrees of freedom for the online learning algorithm. The information fed back from 30 to 60 moves selected by the main algorithm was not consistent enough to make a learning algorithm converge in $\mathbb{R}^3$. The current version uses $d = 2$ which is only one degree of liberty (since the

additional restriction $|L'_{d\times 1}| = 1$ applies) and produces online learning in $\mathbb{R}$. This reduction resulted in a measurable increase in strength.

$H_{d\times p}$ and $H^{+}_{p\times d}$ were computed using Matlab for the Lee and Seung algorithm and Statistica for the pseudo-inverse and for verifying direct and inverse transformations of the complete dataset.

3.4.7 Online learning of $L'_{d\times 1}$

The intermediate encoding $(l_1,\ l_2)$ is a linear combination of the variables described in table 3.2 previously scaled to have each variable's $SD\ =\ 1$ over the 110,542 cases. The resulting $H^{+}_{p\times d}$ learned offline is made of:

- l_1 is a measure strength and speed ($strenLibert \times 0.52$, $speedRows34 \times 0.45$) with a smaller amount of territory ($terriInSide \times 0.21$, $terriCorner \times 0.21$) and speed towards the center ($speedHigher \times 0.14$).
- l_2 is mainly territorial ($terriCorner \times 0.66$, $terriInSide \times 0.54$) with *semeai* fights ($strenAdjLib \times 0.27$) and less of others ($strenStones \times 0.05$, $speedRows34 \times 0.04$).

The intention of online learning is to bias the initial non-informative $L'_{d\times 1} = (1/\sqrt{2}, 1/\sqrt{2})$. $L'_{d\times 1}$ is a unary vector in $\mathbb{R}^2$ (i.e., a direction) making the weight of l_1 heavier (influence based style) or lighter (territory based style) than that of l_2. The underlying hypothesis is: The direction performing best in the previous moves of the same game is better than any constant (game independent) $L'_{d\times 1}$ value. The online learning algorithm is only applied at root. All moves i at root for which the M-eval search returned an evaluation are stored together with their $(l_{i1},\ l_{i2})$ encoding. After the MCTS search, the win rates w_i of these moves computed by the main algorithm are used to compare the performance achieved by 3 candidates:

$L'_{d\times 1}$ (the current value, initially $(1/\sqrt{2}, 1/\sqrt{2})$)
$L'^{1}_{d\times 1} = unary(L'_{d\times 1} + (\delta, 0))$
$L'^{2}_{d\times 1} = unary(L'_{d\times 1} + (0, \delta))$

($unary(\cdot)$ is a function that scales a vector so that $|unary(\cdot)| = 1$, δ is a small step in the direction of either l_1 or l_2.)

The final decision is made by a simple greedy algorithm: The v_{ij} values produced by each candidate j for move i are sorted in decreasing order with their corresponding win rates w_i. Once built, the list is traversed in i. Each time the win rate of one of the candidates is better than those of the other two, a point for that candidate is counted. If the winning candidate has 2 points more than the current ${L'}_{d\times1}$, it becomes the next ${L'}_{d\times1}$ else the current ${L'}_{d\times1}$ does not change.

3.4.8 Results: Positional evaluation

Before analyzing the impact of M-eval as a full board evaluation in playing strength (see 3.4.10), the achievements of the qualities by themselves deserve some attention. In this section we describe some results on the possibility of using the qualities to statistically discriminate between professional and high level amateur games.

The games in our database are from three different sources:

a. Games from the KGS archives filtered by rank of both (*dan* level) players, non-handicap and non-blitz.
b. Games played by professional players in tournaments found in different *go* software collections.
c. Games played by professional players stored in high a quality collection revised by professional *go* teachers.

We wanted to verify if version 1 qualities could be used to identify the level of play between the different collections. To our knowledge, statistical analysis of human *go* games using functions computed from board positions has never been done before.

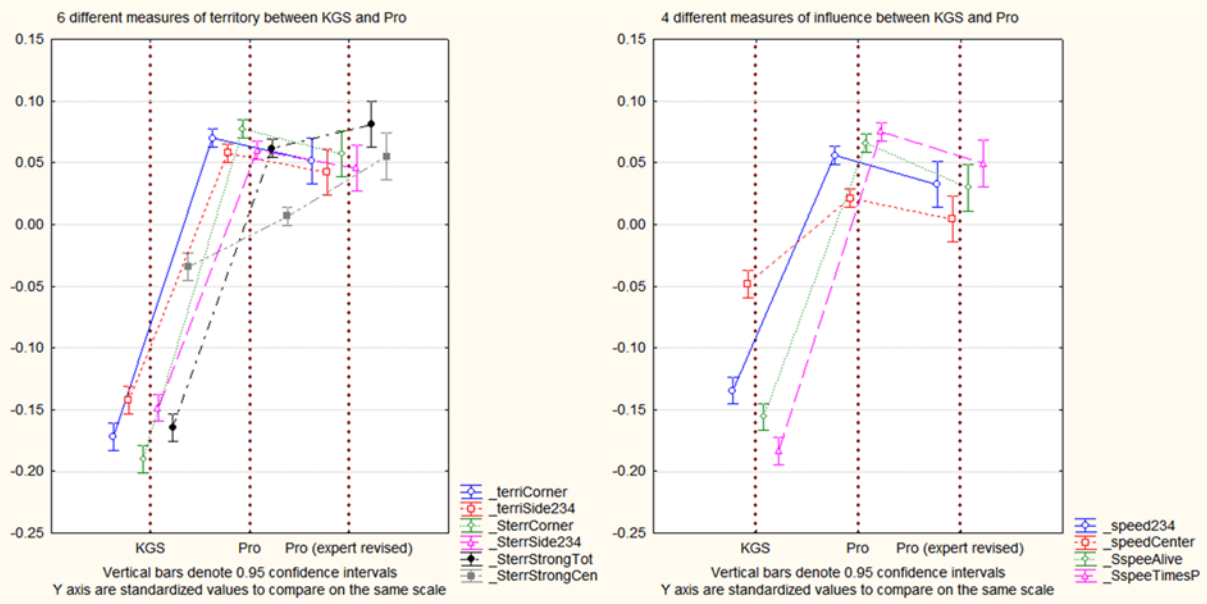

Figure 3.3. Comparison of territory and influence achieved by player level.

Each set of variables (described in the figure) identifies the difference between KGS *dan* level and pro level using MANOVA analysis. Results in figure **3.3** are not surprising: Professionals achieve simultaneously more territory, more influence and more strength than KGS amateurs. It is also interesting to note that the trend is inverted in *semeai* races (see figure **3.4** to the right). Professional players fight *semeai* races by a smaller margin than KGS amateur *dan* players. This is consistent with the superior accuracy of their judgment.

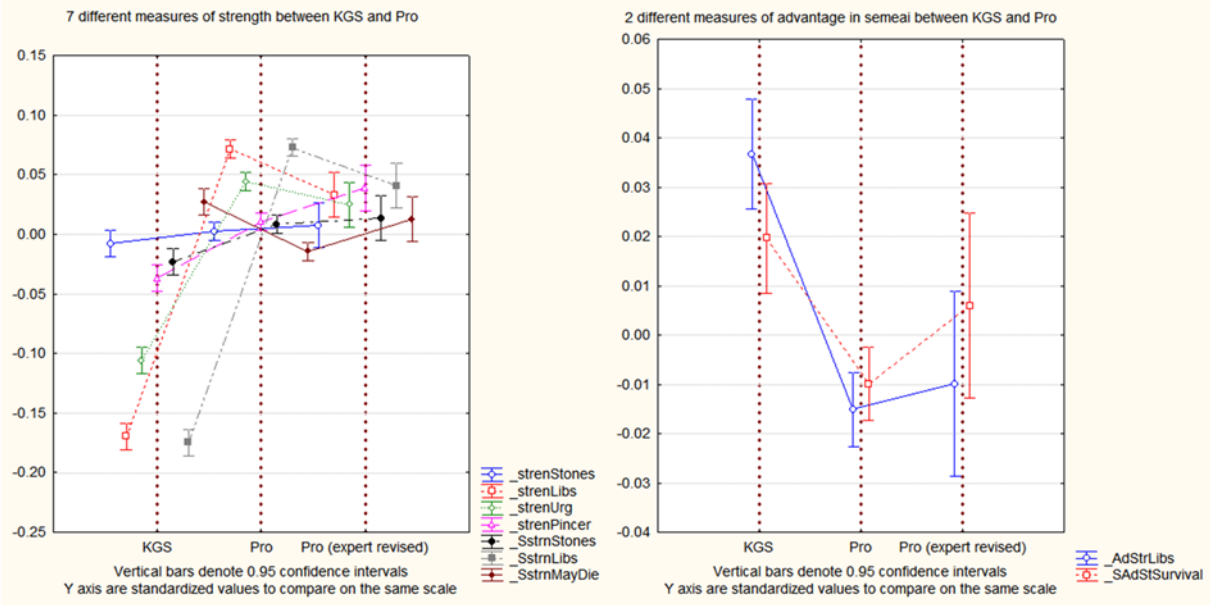

Figure 3.4. Comparison of strength and *semeai* difference by player level.

Figure **3.4** (to the left) shows that the different measures of strength did not work equally well to differentiate by player strength in version 1. In both figures the Y axis measures the standardized values for the variables in each category over the 110,542 cases with the vertical bars denoting 95% confidence intervals. The standardization was only necessary to show different variables (with different scales) in the same graph but it does not influence significance.

3.4.9 Results: Territory estimation

The new territory evaluation functions use a database keeping average estimated territory, its standard deviation and the number of games in which the pattern was seen for a total of 20,201 big corner patterns, 46,979 small corner patterns and 27,881 big side patterns learned from the database of master games. These patterns are multiplied by symmetry, rotation and color alternation. The standard error of the mean is subtracted from the estimation, so returned values can be considered lower bounds for the expected territory in that place found at the end of a master level game when the pattern is seen during the opening.

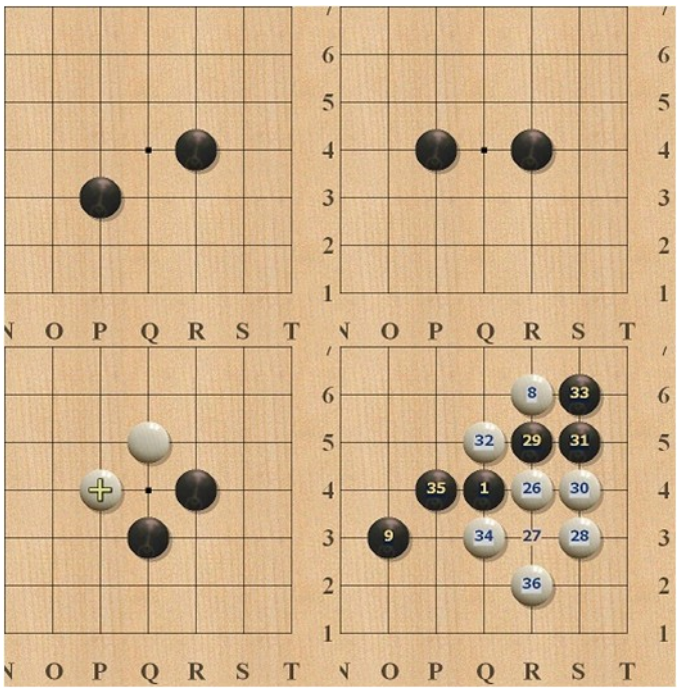

Figure 3.5. Some "top 20" big corner patterns..

Figure **3.5** illustrates some patterns form the list of "top 20" with highest territory estimate. The values given are: territory estimated in cells for each pattern from a total of 66 cells in the corner ± SD. Not surprisingly, (top left) the *shimari* is top 1 worth 45.9 ± 7.46. This means: The black ownership over all evaluated games in which the pattern was found was of 45.9 ± 7.46 points from a total of 66 possible points with the colors shown in the figure. The big *shimari* with the stones in *ogeima* (stone p3 at o3) is worth 43.4 ± 7.73. Top right stones in *ikken tobi* are worth 45.0 ± 7.19. Bottom left, despite the two white stones, the corner makes 45.1 ± 7.74. Not just simple shapes are evaluated. The database includes many complex patterns seen in *joseki*. Bottom right, the *joseki* pattern was seen 34 times in the database and its value is 44.5 ± 9.6. (*Note:* The numbers on the stones are just the move numbers at which the stones were played in the example. The pattern itself is independent from the order in which the stones are placed.)

From the 87 cells in the "big sides", figure **3.6** shows some of the simpler patterns among the "top 20" with highest territory estimate. The left pattern has a value of 55.9 ± 9.69 and was seen 39 times. The center pattern 55.9 ± 8.60 was seen 81 times and the right pattern 55.18 ± 9.85 was seen 264 times.

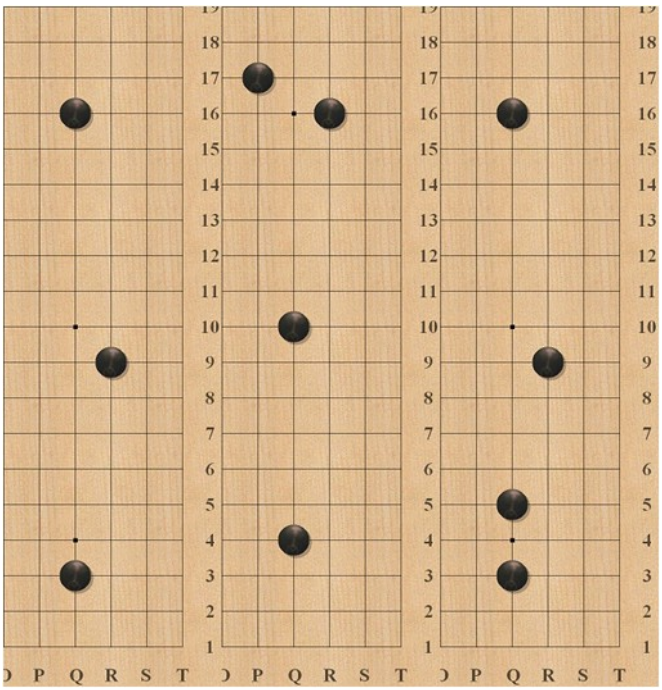

Figure 3.6. Some "top 20" side patterns.

Finally, figure **3.7** shows the "top 5" patterns in the side in order. Note that the *Chinese fuseki* (the very common pattern shown on the left of figure **3.6**) is the 4th best pattern overall, but all patterns achieving more territory also contain more stones. This should not be considered evidence that the Chinese *fuseki* is the best 3-

stone pattern because the result is biased by the fact that it is played frequently by high level players. It is also worth noting that the 2nd best pattern (for black) includes a white stone. We conjecture that this is because the pattern is seen frequently and the three adjacent black stones enclose a rather safe territory. Black is probably not willing to trade it, given his investment (3 stones), while other less concentrated patterns are subject to a wider range of possible outcomes.

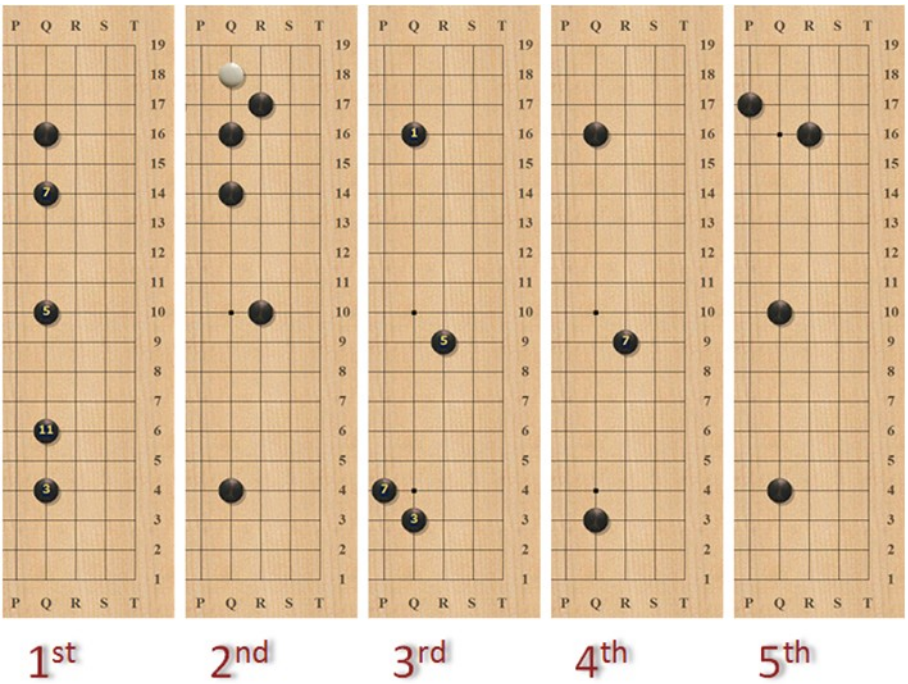

Figure 3.7. "Top 5" side patterns in order.

It is also worth mentioning how we finally changed territory evaluation to estimation by regression. Once we have identified the frequently seen patterns, we can look for the outcome of all games including each pattern. And since we have evaluated each game outcome in terms of territory for each color, we have a good estimate of what each pattern may result when played by strong players, in any case, stronger than the engine. We could also think that the parts of the board not containing known patterns are a problem, but since the moves studied in the evaluation are generated by the patterns, they frequently result in another known pattern whose territory is also estimated. What M-eval does is analyzing these changes in known estimated territory rather than trying to estimate the absolute value.

3.4.10 Results: Overall program strength

Version 1 M-eval was tested with 220 non-handicap 19x19 games (110 with M-eval as black, 110 as white) with *komi* = 7.5. These games were played up to move 50 using the GoKnot/QYZ engine. At that time the program strength in 19x19 was around 15 *kyu* KGS [62]. The games used single core search based on fixed number of iterations rather than time settings. The M-eval program scheduled 4×2500 iterations for M-Eval search followed by 15,000 iterations of MCTS (25,000 in all) while its non-M-eval opponent played 25,000 simulations MCTS. The final positions were evaluated using the public MoGo binary [42]. MoGo is a top MCTS engine and the first program to defeat a professional player in 9x9 *go*. MoGo evaluated each position five times, with the initial *komi* and with a $(4, 8, 12, 16)$ extra points advantage for the non-M-eval version. Since M-eval was only used in the first 50 moves at the time, the intention was to measure the expected positional advance achieved at move 50 rather than its influence on the final outcome of the game. We also considered that the judgment by the best available independent program at the moment of the study (2009) is a more accurate measure of the performance of M-eval than playing the game to the end. Additionally, this evaluation can be repeated with different *komi* values and picture the territorial advantage.

Table 3.3. Results of version 1 evaluated by MoGo in 2009

Komi **(7.5 for white + value against M-eval)**	**Number of wins (from 220)**	**% of wins**	**95% CI**
7.5 + 0	176	80.0%	74.21% — 84.74%
7.5 + 4	164	74.5%	68.39% — 79.84%
7.5 + 8	144	65.5%	58.94% — 71.42%
7.5 + 12	128	58.2%	51.57% — 64.51%
7.5 + 16	112	50.9%	44.34% — 57.45%

Note: Results of self play M-eval vs. non-M-eval played until move 50. The resulting boards were evaluated by MoGo. Also, additional *komi* was assigned against the M-eval version to measure the advantage in terms of territory as well as in terms of winning percentage.

The 80% winning percentage at even *komi* shown in table 3.3 corresponds with a difference of 240 Elo points. The territorial advantage (*komi* difference against M-eval that reestablishes a 50% winning percentage) is about 16 points in 50 moves.

Since the version 1 experiments, our MCTS implementation increased its strength, mainly by the implementation of RAVE, progressive widening, proximity heuristics and fine tuning. The engine, at the time M-eval version 2 experiments were done, was already reliable allowing multithreading in long (several days) experiments with no significant interlocking time loss. It still had progress to be made in playout policies (done

in 2012 as described in 4.7) to reach the strength of top MCTS engines with few simulations. But, because of the wide exploration of its playout policy, it was not prune to systematic misevaluation of shapes observed sometimes in other programs. The main drawback was playing strength (75% winning rate against gnugo 3.7 level 10) required many simulations ($\approx 200{,}000$) unlike other engines that reach a similar level with $1/10$ of the simulations or less.

In January 2011 we performed a series of experiments with our most recent isGO including the most recent M-eval implementation on complete games with a fixed number of simulations per move. 250 games with 4×5000 (i.e., 4 threads running 5000 simulations each), 250 games with $4 \times 20{,}000$ and 250 games with $4 \times 50{,}000$ simulations. The last setting was about 4 *kyu* KGS. Games were played alternating black and white.

We did not introduce any penalty for the computational cost of using M-eval. Nevertheless, the absolute times measured in the worst case (when deploying the root node, since moves are generated for the full board and search is deeper) are typically between 20 and 40 milliseconds. After that, M-eval is much faster as it only generates local moves and search is less deep. Deeper than 4, it is not used at all. The total time per move varies from about 2.5 seconds (at $4 \times 5{,}000$) to about 25 ($4 \times 50{,}000$) at the stage when M-eval is used. Later, MCTS gets faster as playouts get shorter. M-eval accounts for not more than 1% of the computing time in the worst scenario $4 \times 5{,}000$ and even less in $4 \times 50{,}000$.

Table 3.4. Results of version 2 playing complete games in 2011

Setting	Wins / losses	% of wins	Elo increase	95% CI for Elo
4x5,000	138 / 112	55%	36.26	-7.26 — 79.80
4x20,000	157 / 93	63%	90.97	46.19 — 135.79
4x50,000	153 / 97	61%	79.17	34.76 — 123.62

Note: Results of games M-eval vs. non-M-eval, 250 games for each number of simulations (4 cores with 5K, 20K and 50K sims/core). Wins are wins for the M-eval version.

Results of version 2 experiments are shown in table 3.4. The 240 Elo point increase of version 1 was not maintained. Nevertheless, those values cannot be compared directly: Version 1 counted > 50% evaluation by an independent program at move 50, while version 2 counts won games played to the end. A smaller increase is expected in the latter case since the final outcome of the game includes another stochastic process, the fight for the rest of the game. A superior position after the opening is not always game deciding. In the latest version Elo increase is around 80 points which is a significant increase, especially for an algorithm that does not apply during the whole game, but only in the opening. We consider M-eval an essential component of our program

for both computer vs. computer and computer vs. human games and a noteworthy improvement in both style and strength.

Chapter 4. Applying knowledge in the evaluation function

4.1 Evaluation using simulations

MCTS is a tree search and, just like in all tree search algorithms including α—β [82], A* [83] or Df—Pn [84], domain-specific knowledge or generic heuristics suggesting exploration priorities dramatically influence efficiency. This has been analyzed in the previous chapter. Unlike other tree search algorithms, MCTS has a stochastic evaluation function on top of the tree search which plays a key role. At least, when applying MCTS in complicated domains like *go*, uniform random distribution among the legal moves is known to be wrong, both because it misevaluates life and death more than 50% of the time in many simple cases and because it kills living groups without some appropriate rules for handling eye shape and *seki*. This chapter provides insight on the problem of evaluating using stochastic functions and provides a state of the art policy as implemented in a strong computer *go* program.

4.1.1 Stochastic considerations

The first conflict arising when we apply MCTS to a non-randomized, finite, zero-sum, two person, complete information game like *go* is: the value of the game is either a win or a loss, something that cannot be determined for sure (assuming we are not talking about near final positions nor of trivial board sizes). Nevertheless, we are representing it by a "probability of winning" from a leaf node l, p_l without usually emphasizing it is the probability that "some randomized playout policy" applied to both players wins from the

current position l. Furthermore, we are estimating this "true value" of p_l by the estimator $\widehat{p_l}$ observed on a random sample. To simplify the analysis, let's assume the playout policy, unlike the policy described below in 4.7, does not learn online. The probability of winning after a leaf node l, p_l is a constant for each leaf node l and the repetition of wins/losses follows a binomial distribution. Therefore, the proportion of wins following a leaf node l with probability of winning p_l after n random playouts will have the mean $E_l[\hat{p}]$, variance $V_l[\hat{p}]$ and skewness $\gamma_l[\hat{p}]$ following the binomial distribution:

$$E_l[\hat{p}] = n \cdot p_l \tag{4.21}$$

$$V_l[\hat{p}] = n \cdot p_l \cdot (1 - p_l) \tag{4.22}$$

$$\gamma_l[\hat{p}] = \frac{1-2 \cdot p_l}{\sqrt{n \cdot p_l \cdot (1-p_l)}} \tag{4.23}$$

The probability of winning being something other than 0 or 1 implies: it is the sum of most-of-the-time suboptimal decisions. If all the moves in the playout were optimal (minimax-wise), the evaluation would result in the minimax value of the leaf node, being always either win or a loss. The "amount of suboptimality" in each move, as long as it is not biased towards one of the players, can be viewed as a Brownian motion process, whose underlying mathematical model is the one-dimensional Wiener process in discrete time t.

Naming W_t the difference between the minimax value of a leaf node position and the value counted at the end of the randomized playout, being t the length of the playout times some constant representing expected suboptimality at each move, the unconditional probability density function of W_t is:

$$f_{W_t}(x) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t} \tag{4.24}$$

The expectation $E[W_t]$ is zero:

$$E[W_t] = 0 \tag{4.25}$$

And the variance $\mathrm{V}[W_t]$ is t:

$$V[W_t] = E[W_t^2] - E^2[W_t] = E[W_t^2] - 0 = t \tag{4.26}$$

The result of the underlying Wiener process of adding a term W_t to the minimax evaluation of the leaf position is biasing the probability of winning p_l towards $1/2$, assuming it is unbiased. When absence of bias cannot be guaranteed, this may translate to "across" $1/2$ in real applications. The variance of $V_l[\hat{p}]$ does not depend on anything other than p_l as shown in equation 4.22, but p_l diverges from the true minimax value $v_{mm} \in \{0,1\}$ of the position by a factor that increases proportionally (its variance) to the length of the playout.

$$v_{mm} \in \{0,1\} \xrightarrow[V[W_t]=t]{\text{Wiener process}} p_i \in [0,1] \xrightarrow[V_l[\hat{p}]=n\cdot p_l\cdot(1-p_l)]{\text{Binomial distribution}} \hat{p_l} \in [0,1] \qquad (4.27)$$

Equation 4.27 summarizes the framework. We measure $\hat{p_l}$ which is an estimator of p_i in which only the binomial distribution plays a role. But p_i is itself a "wrong" estimation of the position. It is influenced by: the absolute value of the minimax advantage (it is more likely to misevaluate a small advantage than a big one when deciding in terms of win/loss), the length of the playout times some measure of the suboptimality of the policy and, unfortunately, bias in the sense of black/white unbalance.

It is worth noting, for equation 4.25 to be applicable, that bias towards one of the players has to be avoided. This has been stated many times and is also one of the recommendations on playout policies described in [85]. Unfortunately, this is not always possible in *go* where the shapes on the board at root are kept in most of the playouts. It is the shape itself rather than its color which determines how well it is "understood" (i.e., correctly played out) by the policy. This is a source of systematic bias. The groups of one player may have shapes that are systematically over/underevaluated while the other player's groups are evaluated correctly. Also, reducing the impact of bias may be the reason why good heuristics (which should represent moves having less suboptimality) are improved even more when combined with balancing for both players as described below in 4.7.4 and also in Huang et al [86].

It is also important to note that maximizing the size of the wins, rather than the fact that it is a win or a loss, makes the program weaker as stated in the beginning of MCTS *go* by most authors in [19]. Weighting both has been done by [49] based on the argument that big wins are less error prone and also trying to favor "human-style". They found out that the ideal weight for the size factor should be very small (about 2%) being the win/loss factor (98%) the heaviest by far, confirming previous ideas.

Because W_t has a variance that is proportional to the length of the playouts, it is mandatory that when the length of the playout is zero, the position should be scored error free. In many optimization problems (like the Strip Packing Problem or the implementation in 5.6) that is given, but in *go* it is mandatory that the program does not destroy the final position by playing "self-destructing" moves.

4.1.2 Simulation of *Go* games

From the beginning of MCTS *go*, some counterintuitive results have been reported, mainly that stronger playouts do not necessarily produce a stronger MCTS engine, even when comparison is done by equal number of simulations regardless of computing time. In the next section we describe what other strong programs and their authors have stated on simulations. The previous section we have described the underlying Wiener process that helps understanding simulations, two conclusions are very important: convergence to real value of the board when the length of the simulation tends to zero (i.e., understanding eye shape and *seki*) and balancing the number of "smart" moves made by each player as much as possible.

Another key concept is deterministic or near-deterministic response. It is very difficult to understand when it is required. Not implementing it may result in wrong evaluation like a single 4x1 eye being considered dead because the attacker is allowed to play twice in a row as her first move is not answered. Implementing it too deterministically may make the program blind to other fights on the board, e.g., a *ko* fight. In a few cases, we can be sure that a move is always wrong (like playing in the extremity of a single 3x1 eye) and know a way to improve the move, but most of the cases in which the playout policy plays tactical moves, we can easily find counterexamples making the move wrong. There are conflicting interests between giving the move higher priority to "clarify" the evaluation and exploring some other move that may be "just the right move". Of course, unlike misunderstanding *seki*, these tactical weaknesses are to some extent inevitable and will be "ironed out" as the tree search finds the correct sequences establishing the appropriate life and death status of each group. But relying on tree search for correct evaluation (unless the position is already final) strongly reduces the efficiency of the search in terms of simulations wasted, unnecessary nodes allocated and propagating wrong information up the tree until the evaluation is "solved". Completely ignoring answering moves would make the playout so uniformly random that in many cases the correct answer would have a small share in the simulations while its correct evaluation depends on a sequence. This results in the correct move not explored enough in the tree search (when the position is not near final) to expand the search in the appropriate direction making the program systematically misevaluate the position for any realistic number of $(simulations, nodes)$.

The rest of this chapter is about improving simulations with domain-dependent knowledge to reach a strong state of the art *go* engine combined with our original work on learning playouts.

4.2 Playout implementation in other programs

This section includes playout strategies described by the authors of state of the art MCTS *go* engines sorted alphabetically. The list includes the major engines as confirmed by their ICGA tournament achievements, especially those described in scientific literature.

4.2.1 CrazyStone

ICGA tournament results: (Amsterdam 2007) 2nd 19x19, 3rd 9x9

Rémi Coulom, author of CrazyStone describes his simulation policy in [24]. This paragraph summarizes his implementation:

"The pattern system described in this paper produces a probability distribution over legal moves, so it is a perfect candidate for random move selection in Monte-Carlo simulations. Monte-Carlo simulations have to be very fast, so the full set of features that was described before is much too slow. Only light-weight features are kept in the learning system: 3x3 shapes, extension (without ladder knowledge), capture (without ladder knowledge), self-atari, and contiguity to the previous move. Contiguity to the previous move is a very strong feature (gamma = 23), and tends to produce sequences of contiguous moves like in Mogo."

Basically, CrazyStone uses a Bradley-Terry model representing a distribution over the legal moves with each applicable pattern i represented by some value γ_i. The Bradley-Terry model defines how these values are combined when many apply simultaneously to the same move. The γ_i values are offline constants learned from game records with a supervised Bayesian technique. A similar approach, but using local contexts (see 4.3.2) was unsuccessfully implemented in the first MCTS version on the GoKnot engine named QYZ as described in 6.1.3.

4.2.2 Erica

ICGA tournament results: (Kanazawa 2010) 1st 19x19, 3rd 9x9

The program's author Shih-Chieh (Aja) Huang stated *"Erica uses probabilistic simulation completely."*. This is similar to CrazyStone but with different features and also offline learning algorithm. Huang's original work [86] is about the change from *"learning pattern weights with the minorization-maximization algorithm"* to using *"simulation balancing"*. The same paper describes the features:

1. "C*ontiguous to the previous move. Active if the candidate move is among the 8 neighboring points of the previous move. Also active for all features 2-7."*
2. *"Save the string in new atari by capturing. The candidate move that is able to save the string in new atari by capturing has this feature."*
3. *"Same as Feature 2, which is also self-atari. If the candidate move has Feature 2 but is also a self-atari, then instead it has Feature 3."*
4. *"Save the string in new atari by extension. The candidate move that is able to save the string in new atari by extension has this feature."*
5. *"Same as Feature 4, which is also self-atari."*
6. *"Solve a new ko by capturing. If there is a new ko, then the candidate move that is able to solve the ko by capturing any one of the neighboring strings has this feature."*
7. *"2-point semeai. If the previous move reduces the liberties of a string to only two, then the candidate move that is able to kill its neighboring string by giving atari has this feature. This feature deals with the most basic type of semeai."*

4.2.3 Fuego

ICGA tournament results: (Pamplona 2009) 2nd 19x19, 1st 9x9, (Kanazawa 2010) 2nd 9x9, (Tilburg 2011) 3rd 9x9

The policy of Fuego is based on rules that apply following a fixed priority as in the original MoGo work which has influenced many modern programs including our own. This quote describing Fuego's implementation is taken from [49]:

"The playout policy is similar to the one originally used by MoGo. Capture moves or atari-defense moves, or moves that match a small set of hand-selected 3×3 patterns, are selected if they are adjacent to the last move played on the board. Fuego-specific enhancements include a move generator for 2-liberty blocks. If no move is selected so far, a global capture move is selected. If no global capture move exists either, a move is selected randomly from all legal moves on the board. A replacement policy attempts to move tactically bad

moves to an adjacent point. Moves are never selected if all adjacent points are occupied by stones of the color to move, unless one of them is in atari. A pass in the playout phase is generated only if no other moves were produced."

In [87] the policy is detailed as:

"Outside the UCT tree, the play-out phase tries to generate play-out moves based on the play-out policy. The play-out move is generated until a NULL move is generated (i.e., after a pass move was played). The play-out policy generates a move in the following order (from highest to lowest priority):

1. *Nakade heuristic move*
2. *Atari capture move*
3. *Atari defense move*
4. *Low liberty move*
5. *Pattern move*
6. *Capture move*
7. *Random move*
8. *Pass move*
9. *NULL move"*

4.2.4 Many Faces of *Go*

ICGA tournament results: (Beijing 2008) 1st 19x19, 1st 9x9, (Kanazawa 2010) 3rd 19x19

Unfortunately, there is no detailed publication on the internals of Many Faces of *Go* (MFOG) since the engine is based on MCTS. David Fotland, the author of Many Faces has been an outstanding computer *go* author for over two decades. His "classical" engine was described in [60].

In December 2010 he outlined some details about his MCTS engine's playouts in [19]:

"The playouts are pretty heavy, with local responses, hand tuned 3x3 patterns, and moves played with a probability distribution similar to Crazy Stones gamma values, but without the automatic learning. Gamma values are hand tuned. There are rules for not filling eyes, not making self Atari (unless it is a good self Atari),

and avoiding other kinds of bad moves. The eye and self Atari rules are a little different from published methods. There is no tactical look-ahead in the playouts."

4.2.5 Mogo

ICGA tournament results: (Amsterdam 2007) 1st 19x19, 2nd 9x9, (Beijing 2008) 2nd 19x19, 3rd 9x9, (Pamplona 2009) 3rd 19x19, 2nd 9x9

The original MoGo team, Sylvain Gelly and Yizao Wang, pioneered the field of "heavy playouts". The latter being also a strong *go* player designed the shapes of the 3x3 pattern described in 4.4.2 and informally known as "Mogo patterns" in the computer *go* community. In the beginning (around 2006), these ideas appeared counterintuitive also because stronger policies were also tried without success. Quoting [43]:

"Essentially, we use patterns to create meaningful sequences in simulations by finding local answers. The moves played are not necessarily globally better moves. It is not obvious that is more important to get better sequences rather that better moves to make the Monte-Carlo evaluation more accurate. However our experiments showed that the main improvement came from the use of local answers. If the same patterns are used to find interesting moves everywhere in the board instead of near the previous moves, the accuracy decreases. We believe that this claim is not obvious, and one of the main contribution of MoGo."

The complete MoGo policy is further described in [22]:

"Then, since we were not satisfied by the pure random simulations which gave meaningless games most of the time, local patterns are introduced in order to have some more reasonable moves during random simulations. Our patterns are defined as 3×3 intersections, centered on one free intersection, where one move is supposed to be played. Our patterns consist of several functions, testing if one move in such a local situation (3 × 3) is interesting. More precisely, we test if one move satisfies some classical forms in Go games, for example cut move, Hane move, etc.

Moreover, we look for interesting moves only around the last played move on the Go board. This is because that local interesting moves look more likely to be the answer moves of the last moves, and thus local sequence appears when several local interesting moves are tested and then played continuously in random simulations.

We describe briefly how the improved random mode generates moves. It first verifies whether the last played move is an Atari; if yes, and if the stones under Atari can be saved, it chooses one saving move randomly; otherwise it looks for interesting moves in the 8 positions around the last played move and plays one randomly if there is any; otherwise it looks for the moves eating stones on the Go board, plays one if there is any. At last, if still no move is found, it plays one move randomly on the Go board. Surely, the code of MoGo is actually complicated in details, with many small functions equipped with hand-coded Go knowledge. However, we believe the main frame given here is the most essential to have sequence-like simulations."

4.2.6 Steenvreter

ICGA tournament results: (Amsterdam 2007) 1st 9x9, (Tilburg 2011) 2nd 19x19

Unfortunately, we have no knowledge of a publication detailing Steenvreter's playout policies. Its author, Erik van der Werf, author of the textbook on computer *go* [15], mentioned in [19]:

"The playouts just happened to be something for which I already had my own code. Perhaps it was more important that I had a bit different philosophy on what the playouts were supposed to do than most other people at that time. I wanted my playouts to evaluate positions well (and not necessarily play strong moves). So that's what I optimized them for, focusing especially on errors in human final positions, which was easy because I already had a large collection of scored 9x9 games from my time in Maastricht. As a consequence Steenvreter got most of the nakade and seki issues right long before others figured those things out . . ."

4.2.7 Zen

ICGA tournament results: (Pamplona 2009) 1st 19x19, (Kanazawa 2010) 2nd 19x19, (Tilburg 2011) 1st 19x19, 1st 9x9

Unfortunately, Zen's author Yoji Ojima, known as Yamato, has revealed Zen's internals only scarcely, although he generously contributed the first collection of test problems designed for MCTS, on which many test collections, including ours, is based. In one of his few contributions in [19] he stated:

"Zen uses sequence-like AND probabilistic simulation. Basically it plays around the previous move randomly like MoGo, and these moves are biased by gamma values like Crazy Stone. I am also trying to use probabilistic simulation on the whole board, but it does not yet succeed. The main problem is how to combine the semeai, seki and useless-move detection with it."

4.3 Board implementation details related with playouts

This section includes some definitions and board implementations details that are used in both the base playout policy and the learning playout policy described below in 4.6 and 4.7. A description of the board system can be found in 6.1.1.

4.3.1 Legality of moves

The board system keeps track of the legality of playing at each empty intersection of the board. It does this by adding or removing the move from the list of legal moves kept in an object that can also return a uniform random move from it. The board system manages move illegality following standard *go* rules (suicide not allowed and immediate *ko* recapture not allowed). Optionally, the board system implements *superko* but limited to the last 8 positions if the board thread is available (only in tree search). This covers the known board repetitions that may come up in serious games (triple *ko* and double *ko* in *seki*), but not all possible cases of *superko*. Also, in playout mode, the system will consider single point one eye filling moves illegal. This rule is a necessary endpoint for the playouts. The standard one point eye shape definition is an empty intersection with all four immediate neighbors either of the own color or off the board. When intersections at distances (±1, ±1) are cut twice or more by opponent stones or once if some neighbors are off board, the resulting shape is not an eye. When these intersections contain stones of the own color (except at most one), the resulting intersection is always an eye. When at least one of the intersections is empty, the board system supports two versions of the rule: the first one (EyePotential enabled) considers empty as potentially of the same color and therefore the shape is considered a one point eye. The alternative (EyePotential disabled) requires that the empty point is itself an eye (and therefore not playable by the opponent) to consider the eye shape complete. Otherwise, the point is playable since the eye is not yet formed. Empirical testing showed the former to be stronger in self play and it has become the default policy.

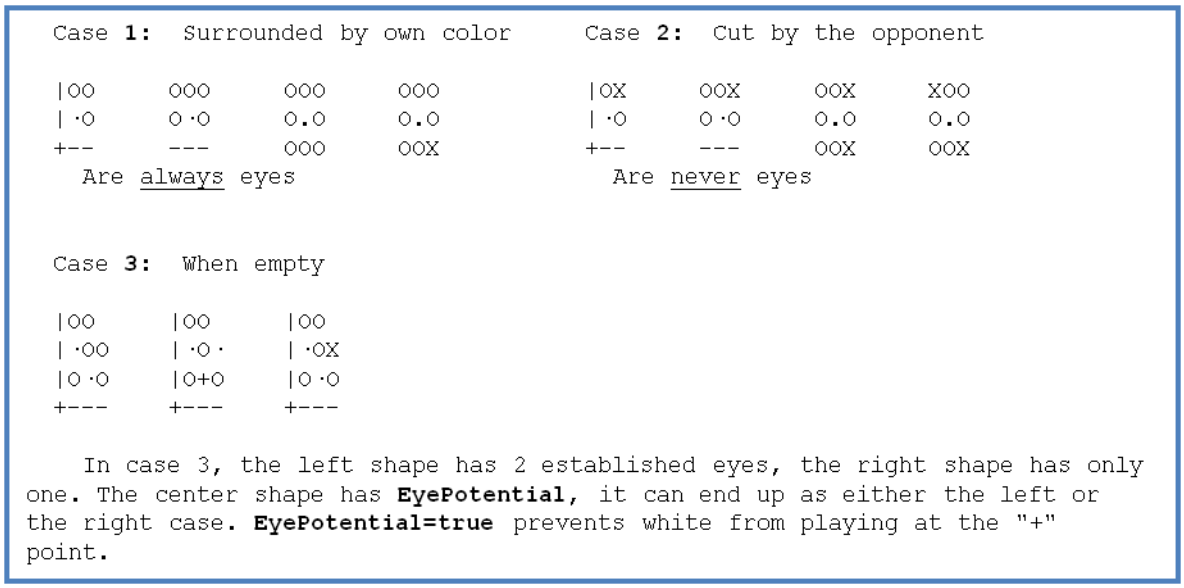

Figure 4.1. Example of EyePotential option.

Figure **4.1** summarizes the idea of EyePotential with an example.

4.3.2 Local context of a move

A local context is a bitmap around each intersection with 4 possible states (empty, own color, opponent color, off board). In the mode used in the playouts, the board system keeps a Zobrist hash of the pattern for each empty intersection (legal or not). The size of the pattern is configurable only at compilation time since each size uses its own set of automatically generated assembly language pattern management routines. The options available are 4, 8, 12, 20, 24, 28, 36 and 40, as show in figure **4.2**. To enable the use of Mogo-style urgent answer patterns the size must be at least 8. The Zobrist hashes computed by the board system are 32 bit unsigned hashes. Due to the large number of intersections considered per second, hash collisions are not "almost impossible" as in other applications, but only limited to some controllable probability of happening. In fact, for practical reasons the implementation only considers a number of bits (12 to 20) from the whole hash value. The whole evaluation by simulation idea is stochastic and accepting the probability of a hash collision around 1/4,096 to 1/ 1,048,576 is not unreasonable.

Figure **4.2** shows the layouts of the bitmaps. Each board size s contains all intersections numbered with integers $\leq s$ in the picture.

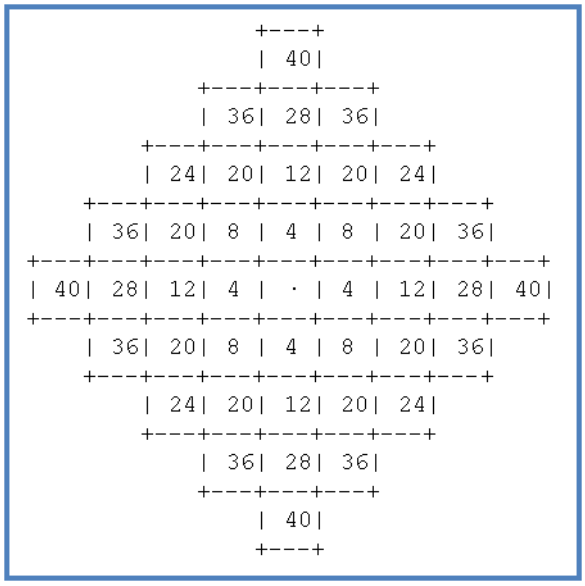

Figure 4.2. All different local context sizes supported by the board system.

Experimental tests showed that larger board sizes are stronger in self play. 24 is stronger than 8, but improvement of 40 vs. 24 was not found statistically significant for 500 games. Besides efficient implementation, larger sizes result in longer computing times around 4% between 24 and 40. The pattern size 24 was used for all the experiments in 4.8 considering proven speed increase is more important than potential but unproven performance increase of larger patterns.

In the following, the local context of a move usually refers to the hash value of the pattern (more precisely to some bits in the hash value) rather than the pattern itself.

4.3.3 Self-*atari* and *seki* handling

As mentioned in 4.3.1, the board system keeps an object containing the list of legal moves and updates legality of each intersection (as stones are placed making adjacent intersections legal/illegal, being created by the capture of groups, etc.) it will notify these changes to the object keeping the list. This object also supports moves being temporarily made illegal. Each time a move is generated (all moves without exception no matter what part of the policy generated it) it is checked to violate some self-*atari* policy $p_{selfatari}$. If this is the case, the move is temporarily made illegal and the policy is forced to generate another move which will also be tested for the self-*atari* policy and either played or made illegal temporarily. Before a move is played, all

temporarily illegal moves are made legal again to keep the board up to date. Different self-*atari* policies are supported, from worst to best:

- p_{ignore} No restrictions. Don't study self-*atari* at all.
- p_{never} Never play a move that is self-*atari*.
- $p_{result\ 3}$ Don't play self-*atari* when the resulting group (including the played stone) has 3 stones or more.
- p_{nakade} Don't play self-*atari* moves when the resulting shape is **not** a known dead *nakade* shape like: 1-stone, 2-stone, straight 3, bent 3 (see remark under figure **4.3**), squared 4, pyramid 4, crossed 5 or bulky 5.

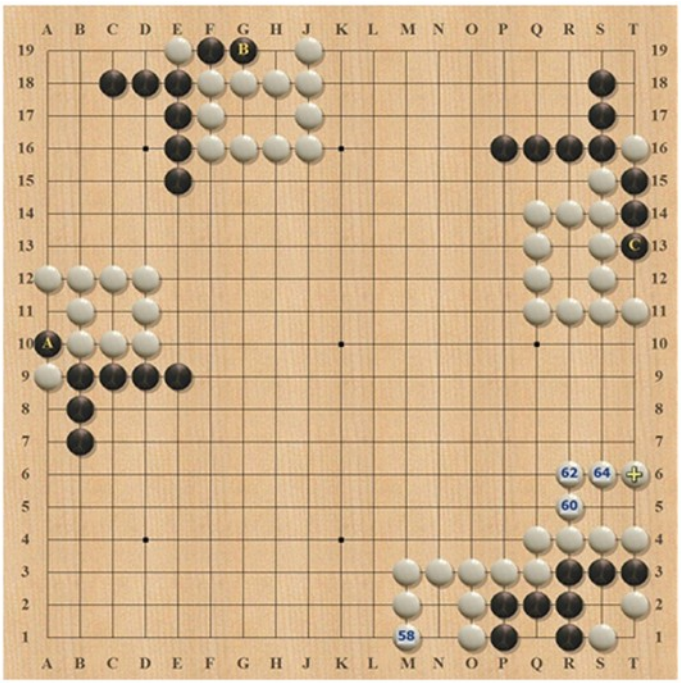

Figure 4.3. A simple rule on self-*atari* moves preserving *seki*.

Figure **4.3** shows the rationale behind the policy $p_{result\ 3}$: One stone self-*atari* moves like A are frequently good to remove eyes. Also adding a second stone as in B is correct. But adding a third one as in C is not, since the opponent can still make the eye. Also, typical *seki* shapes like the bottom right are covered by this rule. Note that the wrong moves at either *s2* or *t1* are only avoided in the p_{nakade} policy because of a trick: The "bent 3" shape only applies if the last stone is one of the extreme stones in the bent 3.

It is also worth noting that not all *seki* destroying moves are self-*atari* moves. At least, when this happens, the growth of the tree should fix the problem making clear what moves are good and bad if the *seki* decides the game. Relying on tree search to fix deficiencies of the playouts has the drawbacks already mentioned in 4.1.2. We will illustrate this with an example:

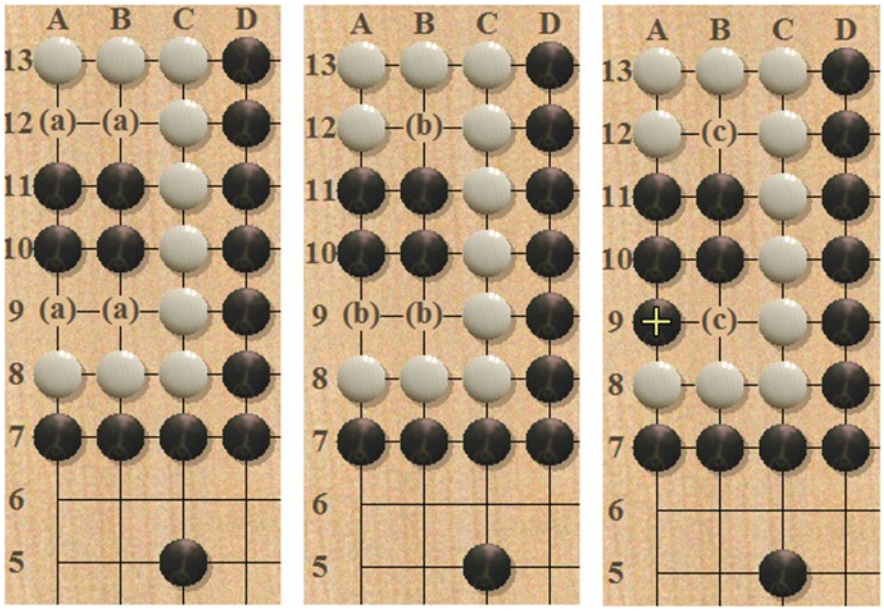

Figure 4.4. Moves destroying *seki* need not be self-*atari* moves.

Because *seki* cannot be identified by static analysis only [88], simple rules will not be able to avoid moves destroying *seki* like white playing (b) in position in the center of figure **4.4**. Therefore, the tree search is necessary to determine the correct life and death status of the white group. To the right, the position is settled, any player playing at (c), which is self-*atari*, kills its own group and that will be avoided by the policy 100% of the time. Before that, playing at (b) is only wrong for white because it kills its own group, but it has no penalty for black other than playing a useless move. Unfortunately, there is no simple rule to avoid white from playing at (b) and that will result in the position being overvalued for black since it ends up sometimes in *seki*, sometimes black kills. Nevertheless, when the tree search reaches (b) the result is correct: white playing is wrong, black playing is *seki* and black not playing is better than black playing. Note that the latter is due to an evaluation error (not playing being overestimated for black by the possibility of white playing (b) by mistake) and even if, in this case, it induces correct behavior, it should ideally be avoided. But, again, there is no simple rule to avoid it to our knowledge. Before that, left of figure **4.4**, playing (a) is possible for both. If black plays first it is always *seki* and will be flawlessly played by the simple policy. If white plays first, all (a) moves result in an equivalent to the center position.

4.4 Successful ideas for simulations

In the previous section we introduced some definitions and board-related details. The "basic toolkit" of *go*-specific simulation heuristics implemented in all strong programs includes besides eye shape legality and *seki* (at least) three more ideas. These ideas are described here as they are implemented in our program. The precise way to implement the features varies from one program to another. Also, a fourth heuristic is mentioned in 4.4.4 although it is not currently implemented since it is an elegant idea that has the potential to better define the influence area of groups even if it comes at the cost of more extensive computation.

4.4.1 *Atari* related tactics

The board system tracks each time a move sets groups in *atari*. It also tracks when groups already in *atari* have increased their size. Both things may happen simultaneously. Additionally, a counter is increased each time any of these events is true and cleared when neither is true. This counter is the "t" described in the base policy used in $p_{AT}[t]$. This helps giving different priorities to ladder related events. It is usual in *go* to play some moves from a losing ladder for a good reason, usually improving the shape of an adjacent group or destroying opponent's eye shape, but it does not make sense to play a losing ladder to the bitter end. Keeping track of the number of consecutive "*atari* tactical" moves allows considering playing some moves without playing the whole sequence. But when enough moves have been played, the ladder must be played out all the times to the end as it must be good for one player and bad for the other. Another way to produce this behavior is to make priority a function of the group size which is the way some programs do it.

The move generation works as follows:

1. Create the set of "atari tactical" moves M_{AT} initially empty.
2. If the board state shows that the previous move has set any groups in atari,
   include all moves that capture any of these groups in M_{AT}.
3. If any groups have just extended from atari,
   if moves capturing the groups exist, include them in M_{AT}
   else, if moves setting the groups in atari exist, include them in M_{AT}
4. If M_{AT} is not empty, play any random move from M_{AT} if $U[0,1) \leq p_{AT}[t]$
   else, return the empty set to the base policy.

   Note: $U[0,1)$ is a uniform random draw in the interval $[0,1)$

Figure 4.5. *Atari* related tactics in pseudo-code.

Note that even if the function supports setting multiple groups in *atari* with one move and both *atari* and extend happening simultaneously, this policy will most of the times return only one move continuing ladders and forcing captures against the corners. Also, when a group can extend making 3 liberties, this policy will not return any answering move since no single move sets the group in *atari* again.

Candidate values for parameters $p_{AT}[t]$ that have been tested always contain some initial probability e.g., $p_{AT}[1] = 0.25$ increased linearly to 1 in 4 to 8 moves being 1 for all moves above that, resulting in only two degrees of liberty: the initial probability and the number of moves required to reach 1.

CLOP optimization [61] returned $p_{AT}[1] = 0.3405$ and $p_{AT}[t] = 1 \ \forall \ t \geq 8$ when the base policy was tuned. These are the values used in the experiments in both the base and learning policies.

4.4.2 Urgent answer to patterns

These patterns, informally known as Mogo-style patterns, are the original ones created by Yizao Wang for [22]. The 8 neighbors of the last move are checked to see if they can be classified in any of the patterns shown in figure **4.7**. The moves are generated as follows:

1. Create the set of "Mogo-style pattern moves" moves M_{MP} initially empty.
2. For each move m in the set of 8 neighbors to the previous move,
   if the pattern around m fits any of the "Mogo-style patterns",
   include all m in M_{MP}.
3. If M_{MP} is not empty, play any random move from M_{MP} if $U[0,1) \leq p_{MP}$
   else, return the empty set to the base policy

   Note: U[0,1) is a uniform random draw in the interval [0,1)

Figure 4.6. Urgent answer tactics in pseudo-code.

The set of all "Mogo-style" patterns is generated using the wildcards described in figure **4.7** and linked to the patterns of 8 neighbors updated by the board system.

```
Patterns including color reversal:

  Are included as they are and also with colors inverted.

XOX    XO·    XO?    ?O?    X·?    ?X?
···    ···    X··    X·X    O·?    w·O
???    ?·?    ?·?    www    ###    ###

Patterns not including color reversal:

XOO    ?XO    ?OX    ?OX    ?OX
···    ?·?    ?··    ?·X    X·O
?·?    ###    ###    ###    ###

Special case (includes color reversal):

XO?                          XO?          XO?
O·?   is pushed excluding    O·O   and    O··  from it
???                          ?·?          ?O?

Legend:

'X' = own color              '?' = wildcard for 'X', 'O', '·'
'O' = opponent color         'b' = wildcard for 'X', '·'
'#' = off board              'w' = wildcard for 'O', '·'
'·' = empty
```

Figure 4.7. All Mogo-style patterns supported by isGO.

CLOP optimization returned $p_{MP} = 0.4812$ when the base policy was tuned and that value is used in the experiments in both the base and learning policies.

4.4.3 Eye shape improvement heuristic

Each time a random move is generated, the candidate position is tested for the number of adjacent empty intersections in their 4 immediate neighbors. When the move has only one empty adjacent intersection, that intersection is played instead if it is legal and has more than one free adjacent intersections.

In all the cases shown in figure **4.8**, a move at A dominates a move at B. For white, the move at A fixes the shape and makes the moves at B one point eyes and therefore, not playable while the move at B destroys the shape. For black the move at B achieves nothing losing one opportunity to destroy the eye shape while the move at A destroys the shape.

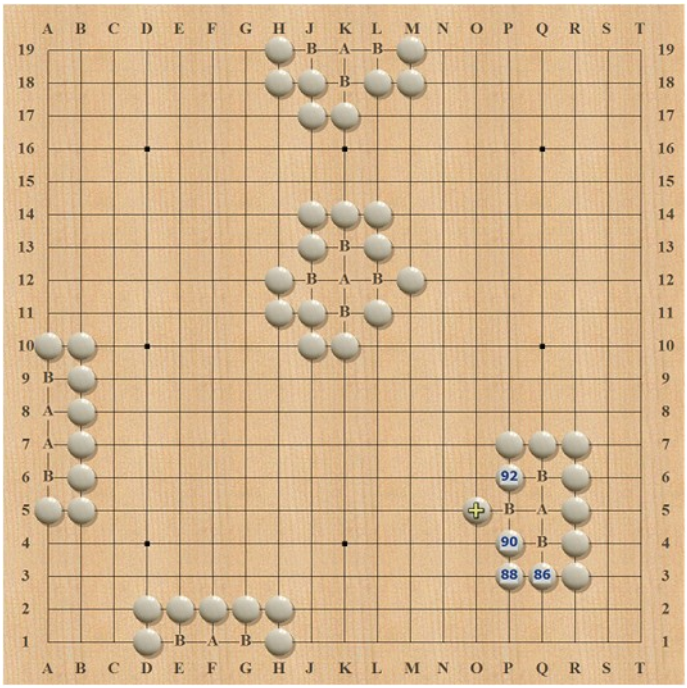

Figure 4.8. Cases for the eye shape improvement heuristic.

The playouts do not understand the urgency of playing A, because that would require a long analysis about what other possible ways of living the group. If the move is urgent or not has to be decided in the tree, but at least, B is always wrong and A dominates B. After A, B will never be played since it becomes an own eye filling move.

4.4.4 Common fate graph distance

It is worth noting that, except for the "*atari* tactics" policy which finds the appropriate *atari* setting, *atari* extending or capturing move(s) wherever they are, all other heuristics rely on the notion of proximity to the last move. This also applies to other move heuristics used by the learning policy. This proximity is usually the nearest 8 intersections, but when increasing size is considered, making the proximity notion match the proximity of groups (at least as they are in the root board) may be a good idea. This can be applied also in the tree search. The MoGo team mentions the idea in [22]:

"We have implemented Common Fate Graph (CFG) [citation] in our program to help the calculation of groups. The method starts from one string and recursively adds close empty intersections and strings close to these empty intersections until no more close strings are found within a distance controlled by a parameter."

They cite [89] which is not easily available. CFG is also described in [90] and above in 3.3.4.

Most authors mentioning CFG use it in the tree search rather than the playouts, but some ideas have been shared in [19] about its use in playouts. The method is computationally expensive (for a playout policy) when computed for each intermediate board, but it can also be used based on pre-computed distances for each two intersections as the chains are in the root board. This version, although generally incorrect, may represent the real influence of the critical groups in root accurately while still being computationally bearable. No implementation of CGF distance has been done so far by our group.

4.5 WLS: Win/loss states

The ideas described in this section have been developed to produce the most efficient possible implementation for our learning playout policy described in 4.7 but are applicable in many fields and no knowledge about the game of *go* is required to understand the entire section. Our original work was first published in [3].

4.5.1 Introduction to WLS

Online learning usually implies storing success rates as a number of wins and visits for a huge number of local conditions, possibly millions. Besides storage requirements, comparing proportions of competing patterns can only be done using sound statistical methods, since the number of visits can be anything from zero to huge numbers. There is strong motivation to find a binary representation of a proportion signifying improvement in both storage and speed. Simple ideas have difficulties since the method has to work around some problems such as saturation. Win/Loss States (WLS) are an original, ready to use, open source solution, for representing proportions by an integer state. The open source implementation can be found at [91].

4.5.2 Definitions

Formally, a WLS with end of scale e is the set S of all proportions $s = n/m$ where $n, m \in \mathbb{N}, 0 \leq n \leq m$ and $0 \leq m \leq e$ over which three functions are defined: $v(s) \to \mathbb{R}$, $W(s) \to S$ and $L(s) \to S$.

The function $v(s)$ is defined over all elements other than $0/0$ and is a real value measuring the evidence that the proportion is above or below a given threshold value. It defines a total order over the set. The functions $W(s)$ and $L(s)$ represent the next state after a win and a loss respectively.

Many choices for the functions $v(s)$, $W(s)$ and $L(s)$ are possible, but we will only consider those fitting the following conditions:

Condition 1: $v(s)$ represents the confidence that a proportion s is above or below some reference proportion s_0 (by default, $s_0 = 1/2$). For this, we use the confidence interval for a binomial proportion using the lower bound (LB) $LB = \hat{p} - CI$ when the $s \geq s_0$ and the upper bound (UB) $UB = \hat{p} + CI$ when $s < s_0$. UB values are shifted by a constant to avoid overlapping between the LB and UB values. $v(s)$ is used only for sorting the set S and therefore, any value avoiding overlapping can be used. We chose the Agresti-Coull interval [92, 93] because its continuity correction provides robust behavior when m is small. Since evidence increases as the number of visits increases: $v(3/4) < v(6/8) < \cdots$ and $v(1/4) > v(2/8) > \cdots$

$$CI = z_{1-\alpha/2}\sqrt{\hat{p}\,(1-\hat{p})/\hat{m}} \tag{4.28}$$

$$\hat{p} = (n + 1/2 \cdot (z_{1-\alpha/2})^2)/\hat{m} \tag{4.29}$$

$$\hat{m} = m + (z_{1-\alpha/2})^2 \tag{4.30}$$

$z_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of the normal distribution for given significance level α.

Condition 2: Updating s after a win always results in an increase of s except when s is already the supremum of S and the opposite applies to a loss. Formally:

$W(s) \geq s \quad \forall s \in S$ and $W(s) = s \Leftrightarrow s = sup(S)$
$L(s) \leq s \quad \forall s \in S$ and $L(s) = s \Leftrightarrow s = inf(S)$
(not including $0/0$)

Condition 3: Let $u(s) = (n+1)/(m+1)$ and $d(s) = n/(m+1)$:

$u(s)\ \epsilon\ S \Longrightarrow u(s) = W(s)$ and $d(s)\ \epsilon\ S \Longrightarrow d(s) = L(s)$

In other words, if the proportion resulting of adding one win or one loss to s is in S, the functions $W(s)$ and $L(s)$ will return that proportion. In other words, only the proportions with $m = e$ require some special attention. These proportions, called saturated, are studied below.

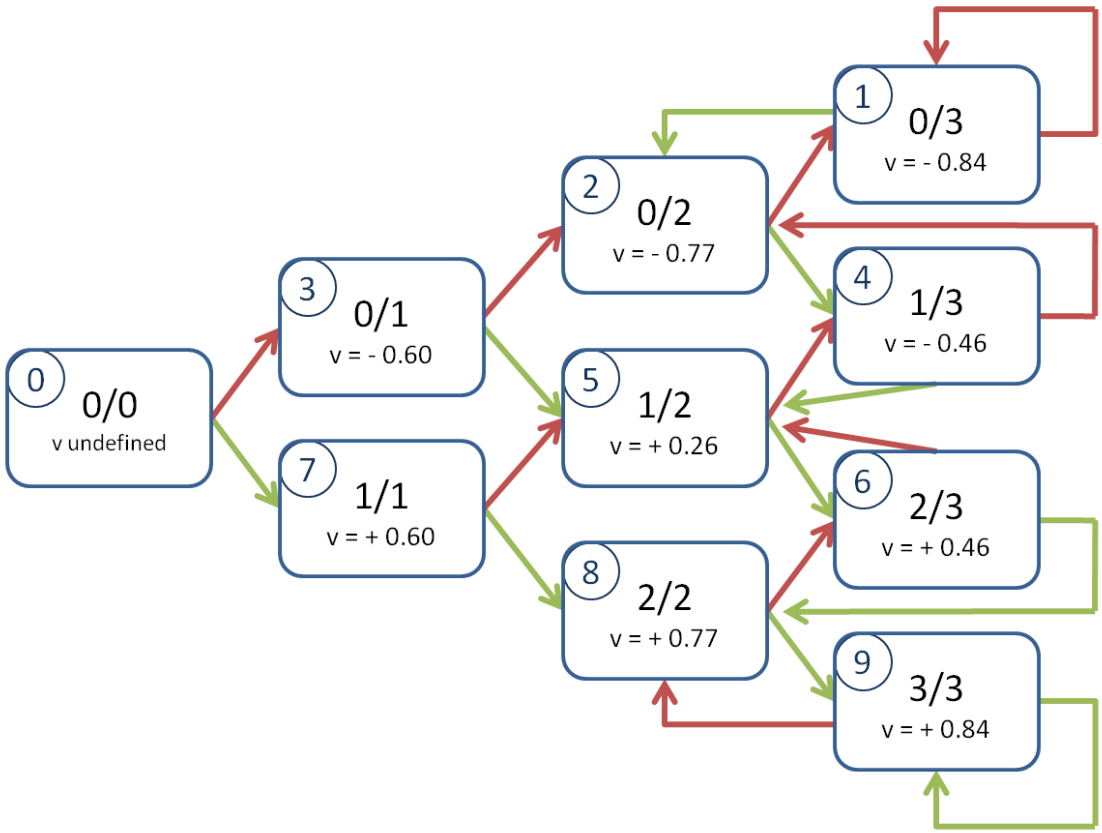

Figure 4.9. Simple WLS with end of scale e = 3.

Figure **4.9** shows a simple WLS with e = 3. Integers in the circles represent the binary value of each state following the order defined by $v(s)$. Green arrows point to the next state after a win and red arrows point to the next state after a loss. Note that the proportions in the last column point backwards following the JPS heuristic.

4.5.3 Implementation in a program

A WLS is an integer that keeps track of the success rate of *(state, action)* pairs. The action is the move played and examples of items defining the state include: the color to move, the last move, the previous move,

the number of liberties of adjacent groups, classified patterns formed by surrounding stones, etc. The possible combinations of input states can be very large. For example, on a 19x19 board, when the state includes 2 moves, the color to move and a local pattern identified in a collection of 100 classes, the resulting number of combinations exceeds 26 million. The amount of information updated is very large as well. If the implementation achieves 15K playouts/sec (a typical performance for an 8 core desktop), estimating 150 moves per playout on average and 30 minutes computing time for the whole game, 4 billion moves are generated during the whole game, each one updating a win or a loss to its combination of states. Therefore, compact representation and efficient integer operation are very important. Any WLS with end of scale $e \leq 21$ can be stored in 8 bits.

During the operation of the program, WLS management requires three basic operations: initializing, updating the state after a win or a loss and checking if the success rate is above some threshold. If it is, the corresponding move is played, possibly with a random choice when more than one candidates are found.

The WLS combination for all possible *(state, action)* pairs is stored as an array $s[\cdot]$ and i_{staction} is the index in the array identifying each pair.

Initializing is done by just clearing the array $s[\cdot]$. Zero represents the $0/0$ state, i.e., the *(state, action)* has not been observed previously.

Updating the WLS after a win or a loss is done via a Look-Up Table (LUT) update. Appropriate LUTs are created during the setup process and stored in some object wt. That class can be found in the source code [91]. After a win corresponding to the state i_{staction} , updating is just $s[i_{\text{staction}}] \leftarrow wt.WIN[s[i_{\text{staction}}]]$ and after a loss $s[i_{\text{staction}}] \leftarrow wt.LOSS[s[i_{\text{staction}}]]$.

Checking if a WLS is above some threshold is just comparing two integer values, since thresholds are converted to integer values in the setup procedure. A typical application will use the binary value corresponding to $1/2$ plus some configurable threshold. In the open source application the binary value corresponding to $1/2$ is stored in the field wt.wls_binThresh_1div2.

4.5.4 The setup procedure

The following pseudocode describes the setup procedure building the LUT tables as implemented in the open source code:

1. Create a table T with all proportions n/m where $0 \leq n \leq m \leq e$. Each item t in T has four fields (n, m, v, i) only n and m are initialized in this step.
2. For each t in T evaluate $v \leftarrow v(n, m)$
3. Assign a value smaller than the smallest v to $v(0,0)$ and sort T in increasing v.
4. Assign an integer number i to each element in T starting with $i(0,0) \leftarrow 0$ and increasing by 1.
5. For each element $t(n, m, v, i)$

   if $u(n+1, m+1, v_u, i_u) \in T$
   then $WIN[i] \leftarrow i_u$
   else $WIN[i] \leftarrow JPS(Won, n, m)$ // Function JPS() is described in 4.5.6.
   if $d(n, m+1, v_d, i_d) \in$ T
   then LOSS[i]$\leftarrow i_d$
   else LOSS[i]$\leftarrow$JPS(Lost,n,m) // Function JPS() is described in 4.5.6.
6. Find the $t(n_t, m_t, v, i_t)$ corresponding to the threshold n_t/m_t and note its integer value i_t for its later use by the program.

4.5.5 The saturation problem

A simple idea for updating a saturated proportion n/e could be:

$$WIN\ [n/e] = s(min(e, n+1)/e) \qquad (4.31)$$
$$LOSS\ [n/e] = s(max(0, n-1)/e)$$

After the number of updated results is larger than e, the WLS behaves just like a counter increasing after a win and decreasing after a loss. This behavior is strongly biased towards 0 and 1. Bear in mind that any proportion above $1/2$ produces more increase than decrease resulting in a state that will be near the top most of the time. And the same applies symmetrically.

To assess how well saturated WLS represent a proportion, we used the following setup: We took a set of 21 WLS measuring the output of 21 Random Number Generators (RNG) generating wins with a probability p_i of winning in equally spaced steps $p_i = \{0, 0.05, 0.1, \ldots 1\}$.

WLS are initialized at $s(0/0)$ and updated a number c of cycles with wins and losses depending on each RNG. The final state was converted back to a proportion $\hat{p}_i$ using the table T described in 4.5.4. $\hat{p}_i$ is an estimate of p_i.

We used two measures to assess the quality of $\hat{p}_i$:

$SD_r = \sqrt{1/(n-1) \cdot \Sigma_i (p_i - \hat{p}_i)^2}$ the standard deviation of the residuals and S_{rc} the Spearman rank correlation [94] between the set of p_i values and the set of $\hat{p}_i$ values, i.e., how well the order of the $\hat{p}_i$ values obtained experimentally represents the order of the original p_i values. We chose two measures, one focused in measuring absolute difference and the other focused in order, since in MCTS order is the most important. A better decision should ideally get higher evaluation than inferior decisions, which makes it explored before them; the value is not relevant.

Each complete experiment, consisting in updating the 21 WLS c times, was repeated 25000 times and the statistics were computed each time. We describe the distribution of SD_r and S_{rc} as $mean \pm SD$ (standard deviation).

Table 4.5. Saturation without JPS heuristic

Number of updates c	SD_r	S_{rc}
c=20 (not saturated)	0.0899±0.0155	0.9624±0.0157
c=200 (saturated)	0.2316±0.0175	0.9170±0.0272

Note: SD_r is the standard deviation of the residuals, S_{rc} is the Spearman rank correlation. Results are obtained for *N=25000* experiments shown as mean±SD.

A nonsaturated WLS of $e = 21$ with $c = 20$ random updates is used as a control reference. Since $c < e$, it does not produce any saturation. Residuals are just those expected by the randomness when $\hat{p}_i$ represents the count of observed wins and losses.

The increased error and the reduction in the Spearman rank correlation shown in table 4.5 both reveal that this idea has a problem with saturation.

4.5.6 Jump-to Past State (JPS) heuristic

The saturation problem is produced because the policy in equation 4.31 makes extreme states $21/21$ or $0/21$ easy to reach by probabilities that are just above or below $1/2$, just because a counter that increases with a win and decreases with a loss will reach any arbitrary number when $p > 1/2$ given enough time. A solution to this is moving the next state to the past to make confirmation necessary before reaching the extreme state again. E.g., if a loss at the state $21/21$ goes to the state, $10/10$ (note that $v(10/10) < v(21/21)$) the WLS needs another 11 consecutive win updates to reach the state $21/21$ again. Also, a win at state $20/21$ should go back to force confirmation with a sequence of consecutive wins before state $21/21$ is reached.

This jump to the past must be longer for the extreme values than for values around 1/2. For the sake of simplicity, we tried a linear model that jumps more as the absolute difference to 1/2 increases. The maximum jump is defined by an empirically tuned constant K.

In pseudocode, the function JPS() used in 4.5.4 is shown in figure **4.10**:

```
Function JPS(won : boolean; n, e : integer)
{
    j = e − round(K · e · abs(n/e − 1/2))
    if (won)
    {
            if (n = e)
                then return the index of element e/e
                else return the index of the smallest n_j/j satisfying v(n_j/j) > v(n/e)
    }
    else
    {
            if (n = 0)
                then return the index of element 0/e
                else return the index of the biggest n_j/j satisfying v(n_j/j) < v(n/e)
    }
}
```

Figure 4.10. Pseudo-code of the JPS heuristic.

We tuned K with the same experiment described in 4.5.5, but with the length of the saturated run randomized to avoid possible fitting to a specific number of times the end of scale is reached. Instead of using a constant $c = 200$, we used a uniform random $c \rightsquigarrow U(150,250)$. The problem starts to disappear as K approaches 1, i.e., the extreme states jump back to $e/2$. The best value found is $K = 1.3$.

Table 4.6. Quality measures of saturated WLS with JPS heuristic

K	SD_r	S_{rc}
0.9	0.1084±0.0147	**0.9742±0.0105**
1.0	0.0964±0.0141	**0.9761±0.0098**
1.1	0.0970±0.0140	**0.9753±0.0095**
1.2	**0.0881±0.0132**	**0.9759±0.0095**
1.3	**0.0817±0.0125**	**0.9759±0.0097**
1.4	**0.0838±0.0129**	**0.9746±0.0102**

Note: K is the constant in the JPS heuristic, SD_r is the standard deviation of the residuals, S_{rc} is the Spearman rank correlation. Results are obtained for *N=25000* saturated experiments with a number of updates $c \rightsquigarrow U(150,250)$ and different values of K shown as mean±SD. Highlighted values show elements behaving better than a non-saturated WLS with $c = 20$.

Note that all results shown in table 4.6 and highlighted are better than the results obtained by the reference WLS with $c = 20$ which is free of the saturation problem. This is consistent with the fact that $\hat{p}_i$ values estimated from approximately 200 RNG draws have smaller variance than those estimated from 20 RNG draws. For validation, identical results within 3 decimal digits were obtained by changing RNG seeds. The change from $c \rightsquigarrow U(150,250)$ to $c \rightsquigarrow U(1500,2500)$ also returned $K = 1.3$ as the optimal value with slightly smaller values of SD_r.

These results reveal that the JPS heuristic works out the saturation problem.

4.5.7 Step response

It is important to note that WLS have the feature of forgetting the past history of the success rates they measure when the probability of a win changes over time. For instance, when we are using results of past moves in a game, before we make a move, the playout policy learns a set of states. Then, a move is made and answered by the opponent. After that, the success rates of all other moves change, but most of the bad moves are still bad moves and most of the good moves are still good moves. Having approximate information may be better than having none. It is a decision of the program author whether to clear past information and start acquiring new more precise but less abundant information, or to keep the old information having more but less

accurate knowledge. Using WLS we have a third possibility: merging past information with new information weighting the most recent events in a controllable way.

From a signal-processing point of view we can consider a WLS as a low pass filter and analyze its step response. Our study is just exploratory and a general procedure for computing the expected bias in $\hat{p}_i$ resulting from temporal changes in p_i is beyond the scope of this study. But, we did empirically establish an interesting result: *The settling time is a linear function of the end of scale* e. This result reveals e as a tunable parameter that may have interesting applications, i.e., instead of always creating WLS with the maximum number of states for a given number of bits of storage, WLS with smaller e may benefit from smaller settling times in applications where that may be an advantage.

In our exploratory study, we only measured the settling time required by a WLS that has been measuring a probability $p = 1/4$ for a long time to reach $\hat{p} = 3/4$ within $\pm 2.5\%$ accuracy when p suddenly changes to $p = 3/4$.

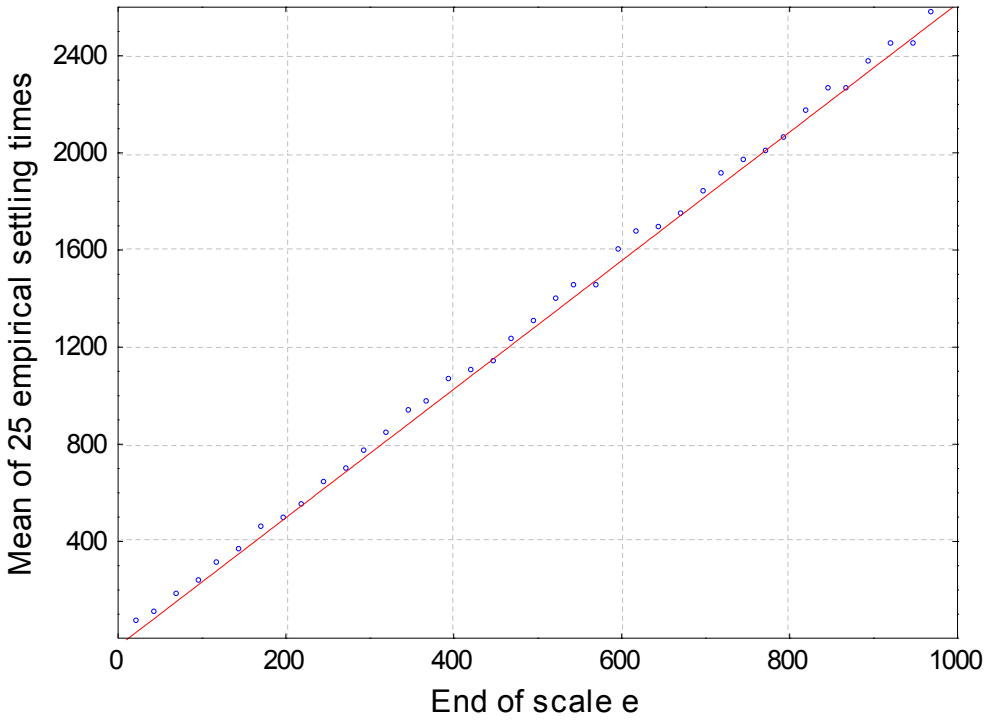

Figure 4.11. End of scale vs. settling time.

We initialized the WLS at the state $1/4$ and generated a number of RNG draws with $p = 1/4$ to randomize the starting point. Then, we counted the number of updates required to measure $\hat{p} = 3/4$ within $\pm 2.5\%$ accuracy when generating updates with $p = 3/4$. The experiment was created for a series of WLS of

different end of scale values e between 25 and 1000 in steps of 25. Each measure was obtained 25 times. A linear regression analysis between the average of the 25 measures and the e values shows a near perfect fit, with adjusted $R^2 = 0.99968$ and p-value $p < 10^{-5}$. The scatterplot is shown in figure **4.11** and reveals a near perfect linear fit between the end of scale values and the mean of the empirical settling times measured from 25 repetitions.

4.5.8 Conclusions on WLS

Efficiency is the key for producing competitive game playing programs. Representing success rates by 8 bit integers instead of complete *(win, visits)* records which result in at least 8 times more space is an improvement when applications may need to store success rates for many millions of possible *(state, action)* pairs. Besides storage, at CPU speed WLS-based implementations outperform *(wins, visits)*-based ones. Updating a WLS is almost as fast as incrementing an integer variable; performing update via a LUT (which can be coded in a single XLAT CPU instruction) is one of the fastest possible operations. If the LUT is in CPU cache memory, which will happen when it is frequently used, the time is similar to updating one or two *(wins, visits)* counters. Furthermore, updating will not happen as frequently as finding new candidate moves. (Even in the minimal case in which only eight neighbors to the last move were considered as candidates, the number of comparisons vs. updates would be 8 to 1.) The latter requires checking if a proportion is above some threshold. Checking that a WLS is above some threshold is just an integer arithmetic instruction, while comparing *(wins, visits)*-based proportions requires computing a floating point confidence interval at the cost of many instructions. If that is not done for the sake of speed, the program will contain inherent flaws derived from unsound direct comparison between proportions based on much evidence and proportions based on just a few observations.

Since WLS is just a brick for building online learning simulation-based applications, prejudice about the lack of importance of doing things "just more efficiently" should be avoided. After all, MCTS is itself a success story about how hardware improvement enabled the possibility to explore ideas that would have been unfeasible one decade before they were implemented. Improvement in both storage and speed pushes the horizon of the "unfeasible" a little further away for new ideas to come. Also, WLS is not *go* specific and can be used in many other machine learning fields.

4.6 The base playout policy

The base playout policy generates moves for the playouts restricted by the rules explained above in 4.3.1 and 4.3.3 following the policy in figure **4.12** (1 is the highest priority, 4 the lowest)

Base policy (main level)
{
1. If a legal "atari tactical" move exists, play it with probability of $p_{AT}[t]$ where t counts the number of consecutive "atari tactical" moves played.

2. If a legal "Mogo style patterns" move exists, play it with probability of p_{MP}.

3. If a legal random move m exists, consider improving it to m' in case m is a shape destruction move. If improvement is possible and legal, play m' else play m.

4. Play a pass move
}

Figure 4.12. "Base" playout policy in pseudo-code.

The simulation ends after two consecutive passes and all stones are considered alive. The parameters $p_{AT}[t]$ and p_{MP} were tuned by self play with the base policy against a fixed reference using CLOP optimization [61]. Both policies (base and learning) use identical values for these parameters although additional improvement of the learning policy could result from using different values.

As an indication of the number of moves generated at each step during the tuning of a base policy in which 6.6×10^{10} moves were generated. The *atari* tactics generated 8.2% of the moves, the Mogo-style patterns 28.1%, 9.4% of the moves were skipped by the random draws selecting not to play the moves although belonging to one of the previous categories, 47.2% of the moves were played at random 3.9% were improved from random moves by the shape improvement heuristic and the resulting 3.1% were pass moves. These values should the considered as just an example taken from tests with board size 13×13 and they may depend on the values of parameters. More such results are shown below in 4.8.2.

4.7 The learning playout policy

This section describes a playout policy representing an improvement over the previous policy known as the "base" policy. The base policy already includes all necessary elements of a strong program, at least stronger than the seminal policy implemented in MoGo which has influenced most strong programs.

4.7.1 The "loose tree" analogy

Rationale: One of the weaknesses of Monte-Carlo methods without a tree, that becomes strength in MCTS, is handling answers to moves i.e., understanding the difference between the board states in which a move is good and the board states in which the same move is bad. MCTS is known to converge to perfect play given some obvious conditions like the policy returning correct results for final positions and the tree not being pruned. Unfortunately, the playouts after a leaf is reached are not getting any benefit from the knowledge in the tree. Ideally, we would like the playout being driven by some of that knowledge at least for some local context as defined in 4.3.2.

The following ideas emerge naturally:

1. Keeping statistics of wins/losses for each move classified by a local context.
2. Keeping statistics of wins/losses for all answers to each possible previous move classified by the local contexts in both the previous and the next move.

Of course, the MCTS tree is based on a sound logic where each node represents a completely defined game state and the "loose tree" idea only aims at stochastically getting things "more times right than wrong". Hopefully, although the idea is a "good intention" and we will not always find moves, that knowledge may drive the playout like the tree does in MCTS, at least using online learned information during the whole playout.

In the worst case, a class in a local context may be the combination of states where the move is good and bad. E.g., a *semeai* with the same local pattern may be decided by the number of liberties at some distant location making the pattern useless. But in other cases, the pattern will decide the difference between good and bad.

Following this "loose tree" metaphor, we name the moves generated by the idea 1 **node moves** because a move associated with a local context from which we keep WLS counters acts like a node in the tree. It is worth noting that the tree node keeps the value of a move associated with a game state and the "loose node" keeps the value of a move associated with a local context that may or may not contain deciding information. The moves generated following the idea 2 are similarly named **edge moves**. Acting like an edge of the tree (an answer to a previous move) with the same differences as in the previous case.

Additionally, since the aim is reducing the number of moves played at random by playing moves that have higher chances of being good, the learning policy is combined with the heuristics already in the base policy.

4.7.2 Node moves: Keeping success of moves by local context

Figure **4.13** shows the basic idea how node moves are played. The empty intersections around the last move are checked (if legal) in the local context as the board stands before the move is played. The move with the highest WLS is played if the value is above threshold.

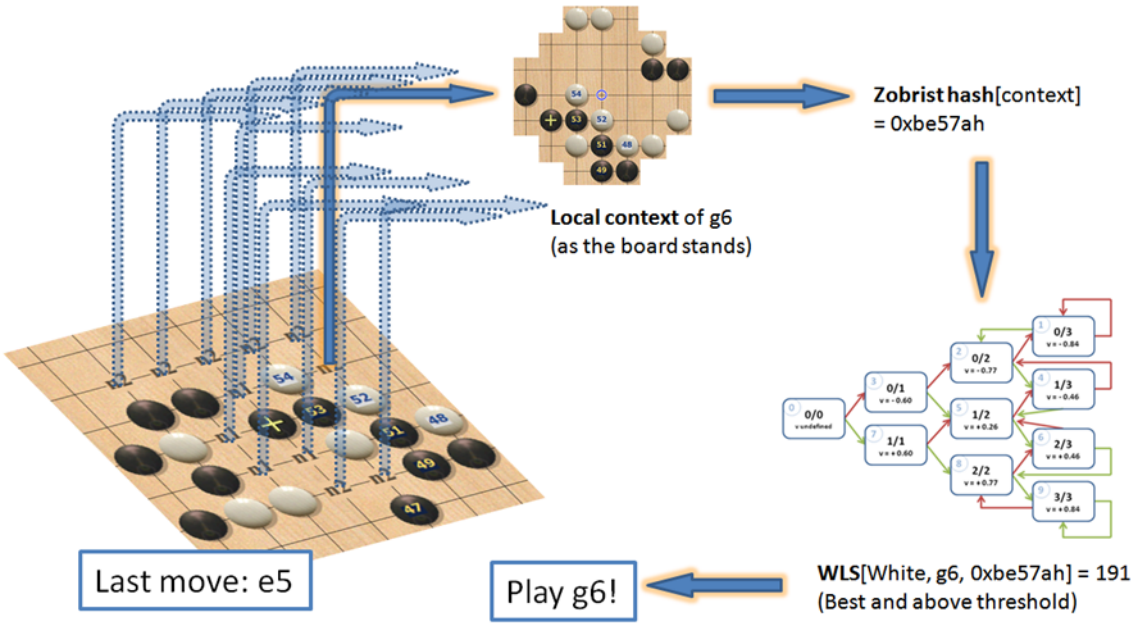

Figure 4.13. "Node move" example.

The knowledge stored for the node moves is a continuous array of WLS states $node\_WLS[]$ that can be thought of as a three dimensional array of 3 indices (i_c, i_m, i_l): black or white, next move, context (an integer in $\{0,..2^n - 1\}$) which is, as mentioned, the lower n bits of the local context of the (candidate) next move.

The threshold is $t_n \pm b_n$ the base threshold for node moves t_n plus/minus an offset as described in 4.7.4. The move played (if any) is the legal move satisfying $node\_WLS[i_c, i_m, i_l] \geq t_n \pm b_n$ with the highest $node\_WLS[i_c, i_m, i_l]$.

The number of neighbors (8 or 20) is configurable and was adjusted in final experiments considering both improvement and speed. It was finally set to 20.

4.7.3 Edge moves: Keeping success of move answers by local context

Figure **4.14** shows and example of edge move generation.

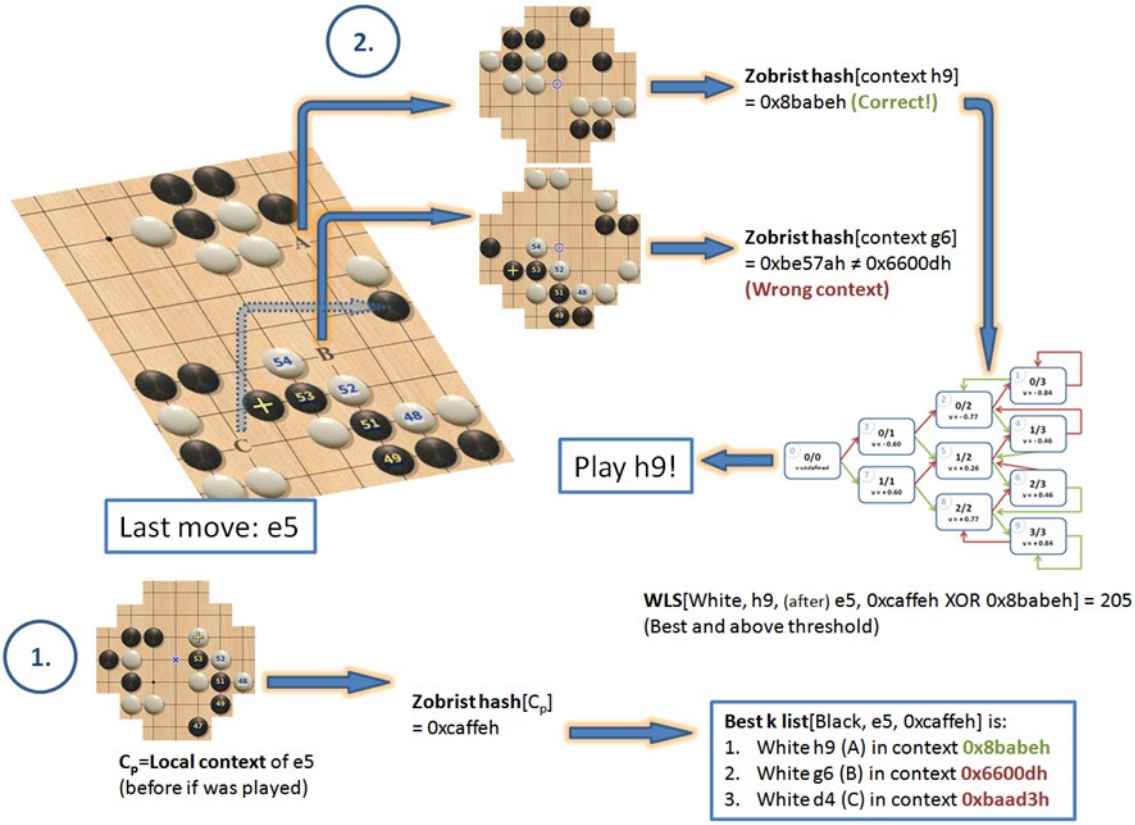

Figure 4.14. "Edge move" example.

The list of "best k" moves in the local context of the previous move and color of the previous player are used to see what the possible answers are and in what context are they expected. If the moves are legal on

current board position and their local context matches the expected local context, the move with the highest WLS is played if its value is above threshold.

The knowledge stored for edge moves is a continuous array of WLS states $edge\_WLS[]$ that can be thought of as a four dimensional array of 4 indices $(i_c, i_{pm}, i_{nm}, i_l)$: black or white (obviously, the previous move will be of the opposite color), previous move, next move, context (an integer in $\{0,..2^m - 1\}$) which is, as mentioned, the lower m bits of the local context of the next move XOR the local context of the previous move.

It would be very slow to search all possible legal moves as answers to the previous move. Since playout policies have to be fast, keeping a list of "best k" answers to each move in each local context (of the previous move) is a solution to this problem. In this case, only the "best k" answers kept of the previous (move, context) are going to be considered. These answers may even be illegal and may probably not fit the local context currently on the board for the next move. It will happen that the "best k" answers do not provide a move even if "somewhere in the edge_WLS[]" a move could be found, but that is compromise accepted for the higher speed of just considering the "best k". Possible values of k range from 4 to 16 and the optimal value was obtained from experimental testing. The value used in the experiments is 16 to favor a higher number of edge moves being played.

The "best k" lists are stored in a continuous array of records $edge\_bestk$ [] that can be thought of as a three dimensional array of 3 indices (i_c, i_{pm}, i_{lpm}): black or white (color of previous move), previous move, local context of previous move (an integer in $\{0,..2^m - 1\}$) which is the lower m bits of the local context on the previous board before the previous move was played.

Precise values of n (number of hash bits used in node moves) and m (number of hash bits used in edge moves) are computed by the program to fit the allocated memory for the knowledge. They depend on the board size s and the assigned size which is a configurable parameter. Note that the array of node-knowledge WLS states will allocate $2 \times s^2 \times 2^n$ bytes times the size of a WLS state (one byte in this case). And the array of edge-knowledge WLS states will require $2 \times s^4 \times 2^m$ bytes times the size of each WLS (also one byte) plus the size required to allocate the "best k" list for each move which is $2 \times s^2 \times 2^m \times sizeof(topNlist)$.

"Best k" records store a list of size k of best moves each containing: the answer move, the next hash and a counter tracking the illegality of each move to purge the list.

The procedure for generating edge moves is described in figure **4.15**.

1. Locate the "best k" list L_{bk} in the array edge_bestk [] knowing the previous color, the previous move and the previous local hash.
2. For each move in L_{bk} check if it is legal on the current board. If legal, clear its illegality counter, if it is not increase it.
3. For each legal move in L_{bk} check if it matches the current local context. If it does, check if its current WLS value is above threshold.
4. Return the move with the highest WLS state matching all conditions in 3, if any.

Figure 4.15. Playing "edge moves" in pseudo-code.

This includes how the illegality counter of the move is updated during move generation. This counter is used for purging the tables from outdated moves. Further details are given below in 4.7.7.

4.7.4 Auto balance B and W move numbers

It was hypothesized that when applied to board positions where one player is clearly ahead, generating moves using a fixed WLS threshold level would produce more moves for one player than for the other introducing bias in the evaluation. Since the learning playout policy generates moves that are on average better than random moves, this would introduce a bias because the system would be "helping the winner". Also, when the winner is getting 75% wins in the simulation, it may consider moves winning over 50% of the times as far from optimal. For the losing player who is winning 25% of the times, moves winning over 50% will be very rare and may be false positives. This player would consider moves winning 30% of the times interesting. The impact of balancing has been established already [86, 95] and we considered using different thresholds for black and white. Also, we use different configurable levels for generating node moves and for generating edge moves.

The condition for playing a node move with index i as black requires $WLS[i] \geq t_n + b_n$ as white with index j it requires $WLS[j] \geq t_n - b_n$. Where t_n is the threshold for all node moves and b_n is the "unbalance" against black, initially zero. The program counts the number of moves each policy generates for black and white. Every 50 simulations adjusting b_n is done the following way:

When the difference in these numbers is bigger than twice the standard error of the mean (the SD is estimated from the binomial distribution) a small correction in ± 1 steps is considered to adjust b_n towards the appropriate direction (increased when the policy is creating too many black moves and decreased when it is creating too few). If one number of moves exceeds the double of the other, a bigger step is considered (4 or 8) to make the initial correction faster. Results logged out by the program show that this simple procedure balances the move numbers well enough resulting in no correction being issued most of the times and corrections being done in ±1 steps when done. Similar parameters t_e and b_e exist for edge moves adjusted by the same procedure.

The feature can be selected or disabled and it was proven experimentally that it results in stronger play in self play experiments. For that reason this heuristic is used in the learning playouts of the experiments.

4.7.5 Definition of the learning playout policy

The learning playout policy is just adding two steps more to the base playout policy.

Learning policy (main level)
{

1. If a legal "atari tactical" move exists, play it with probability of $p_{AT}[t]$ where t counts the number of consecutive "atari tactical" moves played.

2. If the previous move m_{t-1} is not a pass move, the "best n list" of answers to m_{t-1} in the local context of the previous board contains legal moves matching the current local context on the board and their WLS states are above threshold, play the best (in terms of WLS) move fitting these conditions. (Edge moves)

3. If a legal "Mogo style patterns" move exists, play it with probability of p_{MP}.

4. If the previous move m_{t-1} is not a pass move, search the 8 or 20 neighbors of m_{t-1} in the local context of the current board. If their WLS states are above threshold, play the best in terms of WLS. (Node moves)

5. If a legal random move m exists, consider improving it to m' when m is a shape destruction move. If improvement is possible and legal, play m' else play m.

6. Play a pass move

}

Figure 4.16. "Learning" playout policy in pseudo-code

The legality of moves, *seki*, *atari* tactics, Mogo-style patterns everything described for the base policy (even the values of the parameters) applies as described above.

This is the simplest form of learning policy using the heuristics that have been proven to increase strength in preliminary experiments. The thresholds include the balancing mechanism just described. Both edge and node move generators have been tested separately and combined (i.e., disabling step 2, disabling step 4 and both enabled). Both have positive contribution to program strength being the combination of both the best option just as describe in figure **4.16**.

Note that, unlike in the base policy where the choice between multiple moves selected by the same condition is randomized, in the learning playouts the choice is the one with the highest WLS state. As the WLS states are stochastic functions that correlate with the true value of moves, we consider that it should contain enough randomization to avoid deterministic behavior and benefit from choosing the best move more often, although this is not backed up by experimental evidence.

4.7.6 Updating node move knowledge

At the end of each playout, after the board has been evaluated which player won the playout is known. The first n_{upd} moves played in the playout (or the entire playout whatever is smaller) are updated by adding a win to all the moves of the winning player in the context that was on the board before the move was played and a loss to all the moves of the opponent. The parameter n_{upd} is based on the assumption that the first moves in the playout have more influence in the outcome of the game than the final moves. It depends on the length of the playout (which depends on the board size and the current move number). Larger values should produce more information updated, smaller values should produce higher quality of that information. It may also make sense to test values that depend on the board size and the move number at root as a rough estimation of the expected length of the playout. We did not intensively test this parameter. It was set to 80 by CLOP optimization from a categorical variable with 4 possible values: 48, 64, 80 and 96.

The entire array $node\_WLS[]$ has to be cleared at least when a new board is defined. We decided to keep the values from one move to the next without actually testing which of the two options is best: Clearing it at each new move to keep most up to date information or keeping the previous states to have more information. Since the difference is not expected to be big, testing this experimentally may require many resources that were used for more extensive parameter tuning and the experiments described below.

4.7.7 Updating edge move knowledge

The procedure updating edge move knowledge is similar to the procedure updating node knowledge but it is done in move pairs from which neither is a pass. The winner is considered as the answering color, not the previous move color. I.e., if the playout was won by B, the pair of WB will be updated as wins and the pairs of BW as losses. The same n_{upd} parameter used for updating node moves applies to this case.

For each BW and WB pair of moves not including pass moves:

The "best k" list is purged. If any element has an illegality counter above the parameter t_{ill}, the entry is purged. This is intended to allocate room on the list that may be used by old moves that are no longer legal (because maybe a solid group was formed at that spot). After the purge, the lowest WLS in the list is noted to determine the level in which new elements can "reach the top k list".

The array $edge\_WLS[]$ is updated like in the case of nodes (but with 4 indices) and the resulting states are checked if above the threshold for entering the "top k" list. If above, the new element enters the top list replacing the previously lowest WLS found. A new lowest element and threshold is identified.

4.8 Experiments with learning playouts

All experiments in this section compare the performance between isGO (see 6.1.4) using the **base playout policy** described in 4.6 and the **learning playout policy** described in 4.7. In all cases, all other program settings are identical and the engine always does a fixed number of playouts per move. Since the learning playout policy is somewhat slower, we also estimate what improvement should be expected if identical CPU allocation had been used instead of same number of simulations.

It is standard and recommended practice in computer *go* to make comparisons with equal number of simulations. Mainly, because it prevents noise induced by different workloads and other CPU consuming processes running during the experiments and because it emphasizes priority on the science (finding out what works) rather than on the technology (finding the fastest possible way to implement it). Nevertheless, speed

considerations should never be disregarded and, whenever possible, results for equal CPU assignment should be estimated. The method is described below.

Other possible objections regarding the experimental setup typically include:

a. Self-play is not a good measure of improvement.
b. Improvements do no scale as the program gets stronger.
c. The number of games is not enough to draw conclusions.

Regarding objection a, we have included experiments against a fixed reference program, Fuego [49] as it is the strongest reliable free independent program. We also evaluated Pachi [50], but it was not stronger for small numbers of simulations (E.g., 1000), was more memory consuming and we were unable to fix stability issues of its Windows version. Since these games are slower and the experiment requires twice as many games, we conducted them only for board size 13x13 (as a compromise in game duration between the fastest 9x9 and the slowest 19x19) and for just one number of simulations resulting in not too much advantage for Fuego. We conducted 300 games between Fuego and isGO(base) and 300 games between Fuego and isGO(learning). Neither program has an opening book for board size 13x13.

To avoid objection b, we did two extra experiments with 4x8,000 and 4x32,000 simulations in board size 9x9 in addition to the 4x2,000 experiments in all sizes. With this board size and number of simulations, the engine is high *dan* level. On stronger hardware, Fuego has won exhibition games against Taiwanese pro players. (These were mostly handicap 19x19 games, but also won one even 9x9 game and lost one even 9x9 game against the Taiwanese professional player Chun-Hsun Chou 9p in 2009 [96].) We do not have information about the number of simulations used in those games. In our experiments, Fuego is stronger than isGO and the number of simulations used by Fuego was 4x1000 against 4x4000 for isGO. Since, isGO's implementation is somewhat faster than Fuego's, even for learning playouts, in terms of CPU allocation (for the experiments in learning playouts), Fuego's games took 30.3±9.9 seconds versus isGO's games taking 110±23.5 seconds, resulting in a x3.6 factor more CPU allocation to isGO. Also, assuming a 70 Elo point improvement per doubling and the 85 Elo point advantage for Fuego in the experiments, this engine should be around 220 Elo points from the Fuego version winning against the Taiwanese pro on same hardware and same time settings. We performed 200 for 4x8,000 and 200 for 4x32,000 self-play isGO(base) vs. isGO(learning) games in 9x9 to verify that the superiority of learning playouts over the base policy is confirmed (and even increases) at stronger levels. Also, it is worth noting that isGO does not have a 9x9 opening book, this may be

a drawback against other programs, especially with few simulations, but it is desirable for experiments on playouts where using an opening book would be objectionable.

And to assess statistical soundness (objection c), we performed 3 times (board sizes 9x9, 13x13 and 19x19) 1,000 games (600 for 19x19) at 4x2000 simulations to base our claims on narrow confidence intervals. For all results we include winning percentage including 95% confidence intervals, Elo increase including 95% confidence intervals and the Elo increase expected with equal CPU allocation rather than same number of simulations.

This estimation is based on a separate experiment for measuring the improvement of the base policy when the number of simulations doubles. The argument is simple: If the base policy had been assigned the same time used by the learning policy, rather than the same number of simulations, it would have done more simulations. The number of extra simulations is taken from the times used. It ranges from 37% more (in the Fuego experiments) to 59% more (in the slowest 9x9, 4x32000). Each experiment is assigned its individual time factor. It has been established [97] that, at least for a wide range of simulation numbers, program strength increases almost linearly for each additional doubling. We measured this improvement in an independent 11x11 experiment (rounded average of the board sizes used) with 600 games played by the base policy against itself doubled from 4x1000 to 4x2000 resulting in a 480 to 320 win for the doubled version. This is a 70.44 Elo improvement (95% CI is 45.83—95.05). We used this 70 Elo point per doubling improvement for estimating the equal CPU allocation improvement in all experiments by subtracting the resulting improvement expected from the extra time used. E.g., for a 38% extra time, $70 \cdot \log(1.38) / \log(2) = 32.5$ Elo points were subtracted. This is overcompensation since it is known that self play overestimates the improvement gained form doubling the number of playouts. As confirmed by the isGO vs. Fuego experiments, playing the base policy vs. the learning policy does not count as self play since both policies are really different, but estimating the improvement of the base policy based on self play is, as mentioned, an overcompensation.

4.8.1 Parameter values used in the experiments

Table 4.7 shows the values of the parameters described above used in the experiments. Note that common parameters were tuned by testing with the base policy and should be near optimal for it. The learning playout policy could further improve form different settings for these parameters, but we did not test that. We focused on studying the improvement gained from combining the two algorithms (node moves and edge moves) with an existing, more or less state of the art policy.

Table 4.7. Values of parameters used in all experiments.

	isGO(base)	isGO(learning)	Tuning method
$p_{AT}[t]$	$p_{AT}[1]=0.3405$ $p_{AT}[8]=1$ (increasing linearly)	$p_{AT}[1]=0.3405$ $p_{AT}[8]=1$ (increasing linearly)	(1)
k of "best k" lists	n.a.	8	(2)
threshold for edges	n.a.	125/200 (closest integer to this proportion)	(4)
p_{MP}	0.4812	0.4812	(1)
neighbors for nodes	n.a.	20	(3)
threshold for nodes	n.a.	1/2	(4)
neighbors for local contexts	n.a.	40	(3)
Balancing B/W	n.a.	ON	(5)
WLS end of scale	n.a.	21	(6)

Note: Tuning methods: (1) CLOP optimization of the base policy, same values used for both versions. (2) Individually tested, increasing further is slower and does not show measurable improvement. (3) Individually tested, the maximum showed improvement. (4) Only tuned individually in large steps due to the number of games required to assess the winner because of low sensitivity. (5) Tuned ON vs. OFF with clear superiority. (6) Not tested. This is the maximum number for an 8 bit WLS and influence was assessed in the WLS study. n.a. = not applicable.

4.8.2 Number of moves generated at each step

Table 4.8 shows the percentage of moves generated by the different steps in the policies taken form 60 extra games played for the different board sizes: 9x9, 13x13 and 19x19, 20 games each. The base playout policy plays moves found in the previous steps and discarded by the randomization rules, before playing a random move (the 19.04% of moves under "previously skipped").

Table 4.8. Percentage of moves played by each part of the policy.

	Played by isGO(base)	Played by isGO(learning)	learning/base
Total number of simulations	30,251,651	30,160,835	
Total number of moves	7,626,550,755	8,172,206,119	
Atari tactics (%)	7.60 %	6.72 %	0.88
Edge moves (%)	n.a.	3.98 %	n.a.
Mogo patterns (%)	17.66 %	17.43 %	0.99
Node moves (%)	n.a.	33.41 %	n.a.
Previously skipped (%)	19.04 %	4.72 %	0.25
Total before random	**44.29 %**	**66.26 %**	**1.50**
Blind *tenuki* (%)	47.96 %	28.69 %	0.60
Improved *tenuki* (%)	4.40 %	1.53 %	0.35
Pass (no more) (%)	3.35 %	3.53 %	1.05

Note: The complete dataset is based on 20 games in each category (self play 9x9 (4x2000), 13x13(4x2000), 19x19(4x2000)) merged in a single table. The proportions depend on configurable parameters: $p_{AT}[t]$, k of the "best k" lists, threshold for edges, p_{MP}, the number of neighbors for nodes, the threshold for nodes and the number of neighbors for local contexts. The values used are those described in 4.8.1. The values are the total numbers and the mean of the percentages for each board size. n.a. = not applicable.

Even with that rule, it still plays only 44.29% of the moves from the "heuristics" with 47.96% of non improved random moves plus 4.4% of improved random moves. On the other hand, the learning playout policy plays 50% more moves from heuristics than the base policy (66.26%) and it is also worth noting that the number of moves played by recovering skipped moves and improving random moves decreases significantly (19.04% to 4.72% in the first case and 4.4% to 1.53% in the second case). We conclude this is an indicator of the learning playout policy identifying these moves as good and playing them as either edge moves or node moves. Playing less random moves (from 47.96% to 28.69%) is also consistent with the policy being stronger.

4.8.3 Experiments in self play

Table 4.9. Result of experiments isGO(base) vs. isGO(learning)

	Won by learning n of total %	Elo increase (95% CI)	Expected Elo increase for equal CPU allocation
Size 9x9 (4x2000)	663 of 1000 (66.3 %)	**117.6** (94.7—140.4)	(×1.38) **85.1**
Size 9x9 (4x8000)	147 of 200 (73.5 %)	**177.2** (122.3—232.3)	(×1.57) **131.6**
Size 9x9 (4x32000)	148 of 200 (74.0 %)	**181.7** (126.5—237.2)	(×1.58) **135.5**
Size 13x13(4x2000)	685 of 1000 (68.5 %)	**135.0** (111.7—158.2)	(×1.39) **101.7**
Size 19x19(4x2000)	382 of 600 (63.7 %)	**97.4** (68.5—126.4)	(×1.51) **55.8**

Note: The number of simulations is given for each CPU thread times the number of threads. E.g., 4x2000 = 4 CPU threads running 2000 simulations each. The confidence interval for the Elo increase is the Agresti-Coull confidence interval with 95% significance converted into Elo increase. The expected Elo increase for equal CPU allocation is computed by subtracting the expected increase in the base policy by an extra number of simulations proportional to the increase in time (in brackets).

Table 4.9 shows the improvement obtained by the learning playout policy for all board sizes and all CPU allocation times. The improvement even increases as the program becomes stronger. This is consistent with the online learned information becoming more accurate by larger sampling sizes. Since the learning playout policy is automatically learning correct answers to tactical fights like "2 liberty *semeai*" which are not coded in the base policy, a stronger base policy may result in less improvement. Also, the somewhat smaller improvement for board size 19x19 may be due to the number of playouts (4x2000) not feeding back enough information to the WLS arrays. More playouts (and maybe specific 19x19 tuning) should result in more improvement. The expected Elo increase for equal CPU allocation is computed as described above.

Although these experiments are not strictly self-play experiments since the learning playout policy differs enough from the base playout policy to overcome typical shortcomings of self play experiments, we confirmed our findings with experiments against a reference opponent.

4.8.4 Experiments against a reference opponent

Table 4.10. Experimental results in proportion of wins against Fuego

	Won by isGO	Elo increase (95% CI)	Expected Elo increase for equal CPU allocation
isGO(base) (4x4000)	68 of 300 (22.7 %)	-213.2 -(166.0—260.5)	
isGO(learning) (4x4000)	114 of 300 (38.0 %)	-85.0 -(44.4—125.7)	
Improvement (for learning)	15.3 %	**128.2**	(×1.37) **96.4**

Note: All results are for board size 13x13. The number of simulations is given for each CPU thread times the number of threads. E.g., 4x2000 = 4 CPU threads running 2000 simulations each. The confidence interval for the Elo increase is the Agresti-Coull confidence interval with 95% significance converted into Elo increase. The expected Elo increase for equal CPU allocation is computed by subtracting the expected increase in the base policy by an extra number of simulations proportional to the increase in time (in brackets).

The experiments played against Fuego confirmed the size of the improvement in a series of 13x13 games shown in table 4.10. The number of simulations is 4x4000 instead of the 4x2000 in the previous case to further approach the strength of Fuego. The resulting improvement is 128.2 Elo points for equal number of simulations, consistent with the confidence intervals of the 135 Elo point improvement in the previous experiment. The learning playout version was 37% slower than the base policy version; this has been used to estimate the 96.4 Elo points improvement for equal CPU allocation as described above.

4.8.5 Conclusion about the experiments

Besides the highly positive results obtained from a large sample of 4,260 games (3,000 games in the self play experiments, 600 games against a reference opponent, 600 for measuring the scaling factor and 60 for the descriptive statistics in table 4.8.) resulting in around 100 Elo point improvement even with an overcompensation for the slowdown, and improving with the number of simulations, two other indicators show that the whole idea is working as expected:

1. The learning policy is finding the good moves automatically as shown by the reduction in the number of moves produced by the "avoid shape destruction" step and the lower "recycling" of skipped moves. Of course, playing less (35% of the number played by the base policy) "shape fixing" moves (named *" Improved tenuki"* in table 4.8) is in part a consequence of playing less random moves (60% of the number played by the base policy), but the reduction is much stronger in the former case than in the latter, the explanation being that the learning policy is learning the "good moves" and playing them before considering playing a random move.

2. The balancing of black and white moves described in 4.7.4 is contributing strongly to the strength of the learning playouts version. (A large number of specific experiments would be necessary to assess the influence of this particular feature, but the impression from the parameter tuning stage is that it may be responsible for around 30% of the improvement.) This is an indicator that both the edge and the node moves are better than random moves and producing uneven numbers of moves for each player when the game becomes uneven introduces bias as the leading player gets more help than the one in disadvantage.

It is also worth noting that the policy combines well with any other policy and may be a successful replacement for more complicated tactical *go* specific heuristics. Also, since efficiently implementing (in automatically written assembly language as done in our program) local contexts of moves (see 4.3.2) to get as less as a 37% slowdown while computing 40 neighbor local contexts of all legal moves is a huge software development task, we have decided to make our HBS assembly language pattern management functions free software (described in 6.1.1) to encourage replication studies and implement learning playouts in both research and commercial programs.

Chapter 5. Application of MCTS in human genetics

5.1 Genetics in a nutshell

This description introduces necessary concepts required to understand the basic information flow in biological processes. It is required to describe the datasets and the problems defined in the further sections of this chapter. It is intended to provide the computer scientist with enough understanding of the whole picture to determine what specific mathematical models may apply. For a more biologically accurate description see [98] and for a longer description intended for mathematicians and computer scientists see [99].

Most Biological functions are regulated by **proteins**. Proteins can be depicted as molecular machines or parts of molecular machines that serve a specific biological function. Also, a protein may be an **inhibitor** whose presence disables operation of a molecular machine made of other proteins. In humans about 100 000 proteins and their interactions are responsible of all biological functions and visible **traits**. A trait, also known as **phenotype** is a testable condition of a living being like having blue eyes or being lactose intolerant. Most traits are complex traits, i.e. the result of many biological functions and their interaction with the environment rather than a single one. The precise biochemical explanation of a process, i.e., the understanding of all proteins and the chemistry involved, is referred to as a **biological pathway**.

All proteins are made in first place by linking a linear sequence of elementary "blocks" together and then folding that structure in space resulting in a 3-dimensional molecular machine or machine part. The elementary blocks are named **amino acids** and there are exactly 22 different amino acids. Sometimes amino acids are also

called **peptides** and the linear sequence of many is called a **polypeptide**. This sequence of amino acids forming a protein is also known as the **primary structure** of a protein. The secondary and tertiary structures of a protein define how it folds. The former applies locally and is easier to predict as it results from the shapes and electric charges of the molecules involved, the latter requires understanding of the whole protein as an articulated 3-dimensional structure. A protein usually has more than one possible folding solutions corresponding to local minima of its formation energy function. There is also a quaternary structure which defines how the protein interacts with different or identical proteins to form a complex structure. In this chapter we are only concerned about the primary structure of a protein. Nevertheless, the reader must bear in mind that folding of a protein determines its functionality and the tertiary structure often depends on the presence/absence of other proteins. I.e., folding may be the reason why a protein regulates the action of another protein.

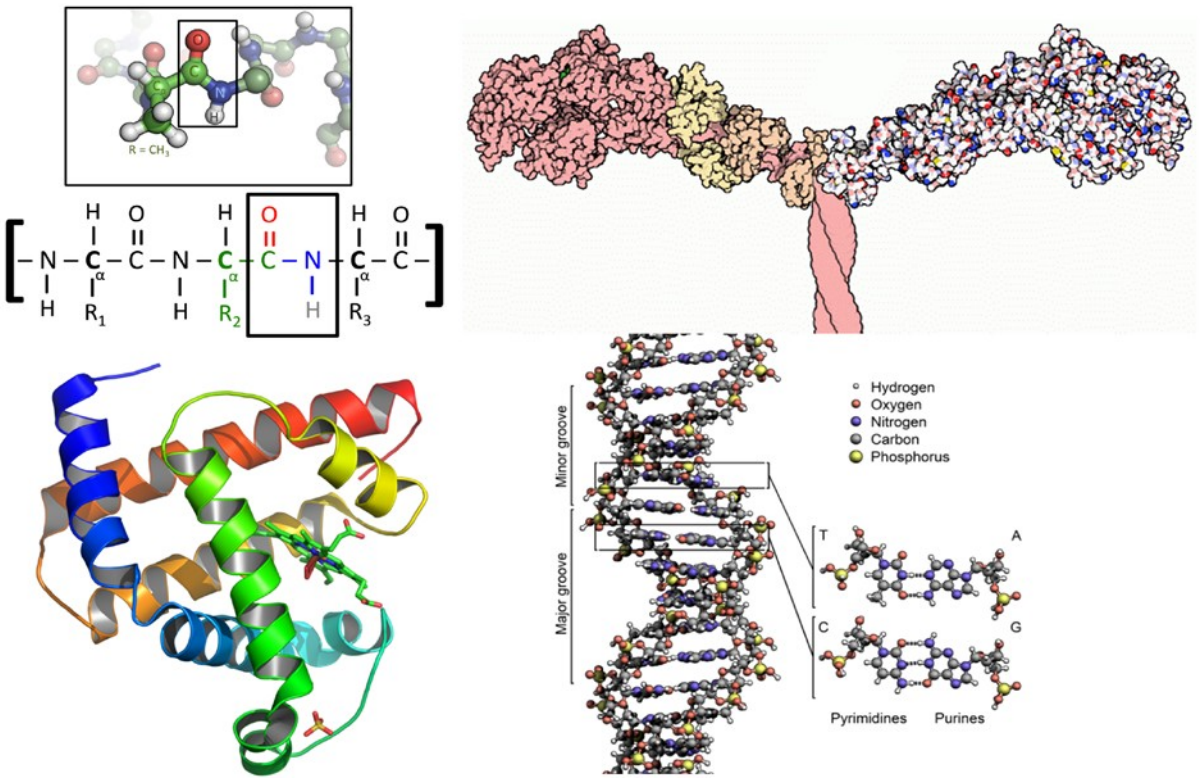

Figure 5.1. Proteins and DNA.

Examples of proteins and DNA can be seen in figure **5.1**. Top left shows the chemical structure of a peptide bond, top right a model of all the atoms in myosin, a protein that has a role in muscle contraction, bottom left a protein as viewed in a protein folding program. Bottom right, part of a DNA molecule.

The storage of information in biological systems, including the storage of information about the primary structure of all proteins, is encoded in **DNA**. DNA (deoxyribonucleic acid) is a macromolecule with a double helix shape. The external part, named the **backbone** chain, is made of sugars and phosphate groups joined by ester bonds and the internal part contains two long sequences of **nucleotides** coding information. A nucleotide is a molecule that is linked on one side to the backbone chain and on the other side to its complementary nucleotide. There are only four nucleotides in DNA, namely A, T, G, and C (initials of adenine, thymine, guanine and cytosine). Current helix models support that the nucleotide A of one of the threads always pairs with a nucleotide T in the other thread of the double helix, and vice versa. The same is supported by C and G. Therefore, both threads, named **strands**, of a DNA molecule code the same information in a complementary way. The backbone does not contain other information than the sense in which the sequence of nucleotides can be read. The two strands containing the complementary sequences of nucleotides run in opposite directions. This is known as anti-parallel coding. A nucleotide is often referred to as a **base** or a **base-pair** because of the double nature of DNA. As bases always come in complementary pairs, one base identifies the pair. E.g., if one strand has the sequence AAGT the other always has TTCA coded in reverse order at the same place.

A DNA molecule of 48 million to 250 million nucleotides in each of its two strands folded with an identifiable X shape under certain stages of the cell cycle is called a **chromosome**. Humans have 24 different chromosomes. Twenty two of them are identical in males and females and are known as **autosomal**. Autosomal chromosomes, also called **autosomes**, are identified by the numbers 1 to 22. The other two are the X and Y chromosomes. Male humans have one of each and females have two X chromosomes. Chromosomes are found in the nucleus of all human cells. Humans are referred to as **diploids**, meaning they have two copies of each chromosome inherited by **recombination** of analogous chromosomes of both parents, with the exception of the Y chromosome in males which is a copy of the father's Y chromosome since the mother has no analogous chromosome.

A **gene** is the necessary information to code the primary structure of a protein. The process converting a sequence of bases in DNA into a sequence of amino acids forming a polypeptide has two phases known as: transcription and translation. **Transcription** is the process of creating a copy of part of the DNA sequence into **RNA**. The RNA is another macromolecule made of a chain of nucleic acids that are complementary of the original A,T,G,C named U,A,C,G (U for uracil). Transcription also involves many phases, among which, **splicing** cuts this molecule and links it again, removing some "chunks", to form the final sequence. **Translation** is the stage converting this modified sequence into the corresponding sequence of peptides.

Because of splicing, the sequence of a gene in the chromosome is not contiguous but is made of **introns** and **exons**. Introns are the parts of the sequence that will be cut off and removed by splicing and exons are the "chunks" of sequence that are found in mature RNA molecules that encode most of the sequence of peptides once translated. Because of splicing and the influence of present/absent proteins, the same gene can code different proteins resulting in around 23 000 genes coding around 100 000 proteins in humans.

Translation reads the sequence 3 bases at a time. A group of three consecutive bases is called a **codon**. There are 64 different codons (the combination of 3 bases) to code 22 amino acids plus some control codons. An example of the latter being the stop codon which ends the translation. In fact, there are several stop codons, one of them is UAG (mnemonic: "You Are Gone"). Also, the same amino acid can be coded by different codons, E.g., The codons UAU and UAC both encode Tyrosine (one of the 22 amino acids). Two codons encoding the same amino acid are known as **synonyms**.

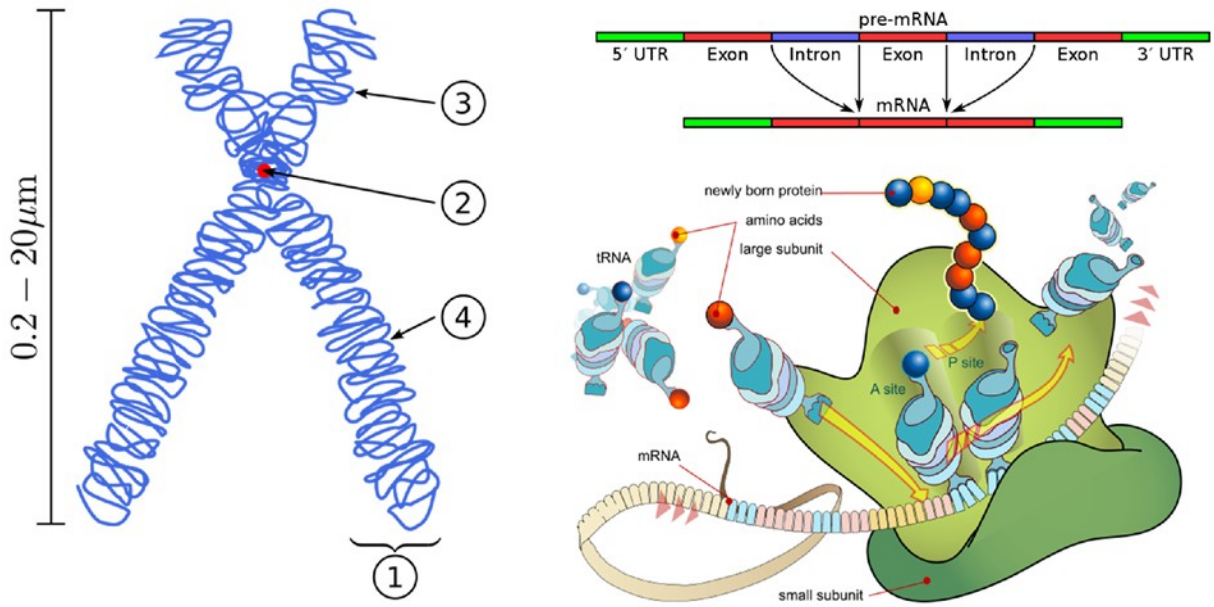

Figure 5.2. A chromosome, splicing and translation.

Figure **5.2** shows a chromosome to the left. In the top right, a DNA sequence is shown before and after the splicing of its introns. Bottom right a diagram shows the translation of messenger RNA and the synthesis of proteins.

The complete genetic sequence of all bases of all chromosomes of a human individual is about 3.2 billion bases long and is named the **genome** of the individual. Except in the case of identical twins, the genome is unique for each individual. Comparing the genome of a large group of human individuals, most of the 3.2

billion of bases are identical in all of them with a small probability (around 1 in 1000) of finding differences at given places. A hypothetical genome sequence containing always the most frequent variant at each place is known as the **reference genome**. The variations found in individuals are several millions in the whole genome and are known as **polymorphisms**. Some of these variations are insertions and deletions of bases altering the length of the sequence, but most of them are just a replacement of one base by another. When that replacement has only two possible options, it is named a **SNP** (single nucleotide polymorphism) *(pron. snip)*.

The point where a SNP takes place (E.g., chromosome 17, base number 13571 of the reference genome UCSC Assembly Mar 2006) is known as a **locus** *(pl. loci)*. To identify loci even when the reference genome changes, possibly resulting in base number changing due to insertions/deletions, an **rs number** is used as a unique identifier for the locus.

Each of the two forms of a SNP (or more forms in other types of polymorphisms) is named an **allele**. Polymorphisms in the human genome are strongly statistically dependent on each other. This is known as **linkage disequilibrium**. As a result of this dependency, a set of 650 000 SNPs have been used to precisely predict the variation at around other 2 million SNPs from the whole genome [100]. Because of that, some sets of alleles at adjacent locations are transmitted together. I.e., some SNP variants are always seen in combination with always the same allele in another SNP. These associated sets of SNPs are known as **haplotypes**.

Since, as mentioned, humans are diploids meaning they have two copies of the same chromosome at any given place, with the exception of the X and Y chromosomes in males and mitochondrial DNA in both sexes; each individual will have two alleles. These two alleles may be identical, in which case the individual will be called **homozygous** for that SNP, or different, (*idem*. **heterozygous**). Because of that, and because the focus of this work is on SNPs with two alleles, each individual will harbor one allele combination out of the three possibilities for a given locus. E.g., when the possible alleles are A or T, the possibilities will be: AA, AT and TT. The type TA is equivalent to AT. These combinations, or any set of them extending to other parts of the genome, are referred to as **genotypes**. Genetic polymorphisms, including SNPs, are also commonly known as **markers** especially when they are part of a model explaining a phenotype (not the actual causal variant of the disease or trait) or in the field of population genetics.

In case a single SNP is responsible of one trait, i.e., If all homozygous individuals with genotype AA at a given locus share the trait Z+ and all homozygous individuals with genotype TT share the complementary trait Z-, the following definitions apply: If all heterozygous individuals AT share the trait Z+, the allele A is said to

be **dominant** to the allele T and the allele T is **recessive** to the allele A. If the heterozygous individuals show an intermediate trait, the alleles A and T are said to be **additive**.

When the parent is homozygous for a given SNP he/she can only transmit the allele that is present in his/her own genome. When the parent is heterozygous, he/she may transmit any of the two alleles to the descendants. The resulting genotype of the descendants will be constituted by the combination of the alleles transmitted by each parent. When a parent transmits an allele at a given locus L_1, the probability that he/she transmits another allele at another locus L_2 (in the same chromosome as L_1) from the same copy of the chromosome used to transmit at L_1 (rather than from his/her other analogous chromosome) is a decreasing function of the distance between the two loci $d(L_1, L_2)$. This is explained by a mechanism that most commonly happen during division of germline cells termed as **chromosomal crossover**. Chromosomal crossover is a known cause of linkage disequilibrium.

Nowadays, a large number of SNPs (e.g. one million) can be detected simultaneously for a given individual DNA sample using a **DNA microarray** (also referred to as DNA chip or SNP chip). A DNA microarray is merely a solid surface about 1 cm^2 with a collection of spots, each spot containing a **DNA probe**, a small DNA chain that is complementary in sequence to a specific part of the genome and that will only give a positive signal if both sequences (the probe and the genome region) are fully complementary. The probe also contains a fluorophore, which is a covalently bonded molecule visible using microscopic fluorescent imaging. Fully complementary matches (i.e., matches with one allele at a particular SNP) will bind strongly each other, while partially complementary (i.e., the probe does not match with the allele at a particular SNP) bind weakly and are washed off by a specific process. The specificity of this process is therefore dependent on the flanking sequence of the SNPs, as each of them has a unique sequence that allows them to be distinguished by the process.

The entire microarray is scanned at high resolution and a microarray processing program generates a file with the genotype calls at each given SNP for all SNPs **genome wide**. In the process, a quality control filter specific of each company and model remove low-confidence calls based on pre-established thresholds. Given the massive number of genotypes to be obtained for each sample, and the number of samples that are commonly used in these studies, one has to anticipate that a number of genotypes (<5%) may be missing from the studies.

The application of MCTS in genetics described in this PhD dissertation uses the GEM (Gene Etiology Miner) libraries, an original work of the author described below in 6.2. These libraries use the following input files:

A **genotype file**: Each row is an individual identified by a unique integer ID and containing a large amount of genetic data. Microarray sizes in the current market usually contain range from 10,000 to 2 million SNPs. The variables of the dataset are the ID followed by the reference sequence of the SNP (identified by rs numbers) that uniquely identify the thousands of SNPs contained in the microarray. Each of these genetic data is a categorical variable with four possible values: RR, RA, AA and missing. RR means the subject is homozygous for the reference allele, RA is heterozygous and AA is homozygous for the alternative allele. The value will be missing if the company's algorithm considers it to be a low confidence call based on pre-established thresholds.

A **phenotype file**: Each row is an individual identified by the same ID as in the genotype file and the other variables are traits of the individual. In general, traits can be sets of continuous and categorical variables. For simplification, in our study we will only consider a single categorical trait variable. In the biomedical field, most studies are referred to as **case-control** studies of unrelated subjects. This means each individual does or does not have the disease under study, the former is known as a case, the latter as a control. This is a particular case of categorical variable in $\{case, control\}$. Our application can also use k categories as in our World 7 experiments shown below in 5.8. Also, more complicated traits (e.g., multivariate traits) can be categorized as a previous step and the genetic factor explaining each category can be looked into as a categorical trait.

SNP functionality files: These files contain a list of all SNPs identified by rs numbers and contain information for:

a. The physical position of the locus (chromosome, position and reference genome)
b. The gene or genes overlapping or nearby that position, which can be retrieved from a gene definition file.
c. Coding details, if available. Only if the SNP at that locus is part of a coding region (exon). In such case, the file also specifies the functional prediction of the nucleotide change because of the SNP: It predicts no change of amino acid (it is a **synonymous** change), it predicts a codon change so that an amino acid would be changed by another (it is a **missense** change), it predicts a codon change in a way that an amino acid is changed by a stop codon, or the other way around (it is a **nonsense** change).

d. Information about established linkage disequilibrium. What other SNPs not present in the microarray can be predicted by the knowledge of this SNP and based on which SNP combination.
e. Information about all the genome wide association studies (GWAS) that mention association between the SNP or the gene and diseases explored to date.

SNP functionality files are compiled from public databases and updated regularly by the GEM system. In the MCTS experiments described in this chapter, SNP functionality files are not used.

5.2 The genetic etiology problem

The word etiology is used in medicine to refer to the cause of disease. Therefore, genetic etiology is about predicting risk of disease based on the knowledge of the genotype of studied individuals. In this context, that genotype represents the genotype composition at the entire genome or most of it, i.e. at genome wide level.

In general, the problem of finding the genotype explaining a given number of traits, possibly using other information such as gene functionality, gene expression or trait networks, is known as the genotype/phenotype association problem. The Genetic Etiology Problem (GEP) described in this thesis is the particular case of finding the set of n SNPs that best classifies the population in k groups. This supersedes the case where $k = 2$ and categories are in $\{case, control\}$ used as a rule in medical studies. The mathematical model used for this classification is described below in 5.5.2. Some caveats must be pointed out first.

a. **Most diseases are complex diseases**. This means that they do not always have the same causing gene (many genes) involved and, most likely, they are not due to genetics alone. Patients are diagnosed with a disease when they match a consensus definition for the disease. That definition is a collection of measurable symptoms, not their causes. Also, genetic association may apply even in cases where we would not expect it (e.g. many diseases classically considered to be due to environmental causes such as infections). For instance, patients satisfying the consensus definition of chronic obstructive pulmonary disease (COPD) are most of the time smokers or ex-smokers. The smoking habit should not have a genetic cause. Therefore, we would not expect a genetic association with COPD. Searching [101] we find out that: *"Alpha 1-antitrypsin deficiency is a genetic condition that is responsible for about 2% of cases of COPD. In this condition, the body does not make enough of a protein, alpha 1-antitrypsin. Alpha 1-antitrypsin protects the lungs from damage caused by protease enzymes, such as elastase and trypsin, that can be released as a result of an inflammatory*

response to tobacco smoke." Except in the case of highly rare diseases, we will scarcely find a genetic cause or a unique gene involved in the explanation of all the cases. Instead, for complex diseases, one should expect to find a couple dozen or even hundreds of associated SNPs constituting risk alleles. Also, bear in mind that because of linkage disequilibrium, alleles associated with risk may not be causal, but just statistically correlated with causal alleles,

b. Since the problem is so loosely defined, **the number of candidate models is very large**. E.g., there are around 2.7×10^{53} sets of 10 markers chosen from 1 million SNPs. The risk of overfitting the model and the risk of false positive associations is huge. This is even worse since, as just mentioned, a sound association may apply only to a small proportion of the cases. This makes standard adjustment for multiple tests (e.g., Bonferroni adjustment) almost useless. In this thesis, we handle this by considering only very simple models and cross validating the classification.

c. Also, caveats apply when **association** (i.e., statistical dependency) is used for classification (**risk prediction**). Strong association does not necessarily guarantee good classification or discrimination ability [102, 103]. The presence of unknown associations may also play a role. [104] states: *" If there are 200 undiscovered locus associations, more than 7% of the subjects who are estimated to have triple the median risk of disease actually have less than the median risk."* This discussion is somewhat beyond the scope of this thesis, but this problem is taken care of using a mathematical model that is a classification method.

d. Underlying biological **pathways** involved in disease pathogenesis are expected to be **very complex**. As mentioned in the introduction, SNPs can modify or completely abrogate the functionality of a protein. Also, a protein may only be functional in the presence/absence of another protein. This would justify mathematical models including the interactions of SNPs. Even with a very small number of SNPs such as 3, the resulting 27 combinations would require minimum population sizes of many hundreds to be able to observe the number of observations expected in each category within a few tenths of individuals. This is feasible in studies with sample sizes around a thousand or a few thousands individuals. Even in this case, the risk of false positives is huge due to the high number of models being tested. Beyond 3 SNPs, mining complex models without strong prior hypotheses is not feasible with realistic sample sizes. This must be accepted as a limitation of data mining without a priori information in such a loosely defined problem. If we consider simplifying the model, even if the model will not represent all possible biological complexity, we obtain models with 5 to 20 SNPs that classify cases and controls as seen in 5.8. This is suitable for an MCTS implementation and is consistent with the **common disease–common variant hypothesis**: A theory that many common diseases are caused by common alleles that individually have little effect but in concert confer a high risk. [105]

e. **GWAS must be replicated**. The NCI-NHGRI Working Group on Replication in Association Studies has made some recommendations in the form of a checklist [106].

Table 5.11. Recommendations on replication in GWAS.

• Statistical analyses demonstrating the level of statistical significance of a finding should be published or at least available so that others can attempt to reproduce the reported results • Explicit information should be provided about the study's power to detect a range of effects • The study should be epidemiologically sound, with careful accounting for potential biases in selection of subjects, characterization of phenotypes, comparability of environmental exposures (when possible) and underlying population structure in cases and controls • Phenotypes should be assessed according to standard definitions provided in the report • Associations should be consistent (within the range of expected statistical fluctuation) and reported for the same phenotypes across study subgroups or across similar phenotypes in the entire study group • Significance should not depend on altering the quality control methods beyond standard approaches that could change inclusion or exclusion of large numbers of samples or loci • Measures to assess the quality of genotype data should include results of known study sample duplicates or publicly available samples • The results for concordance between duplicate samples (if applicable) as well as completion and call rates per SNP and per subject should be disclosed, along with rates of missing data • A subset of notable SNPs should be evaluated with a second technology that verifies the same result with excellent concordance, because no technology is error-free • Associations with nearby SNPs in strong linkage disequilibrium with the putatively associated SNP should be reported (and should be similar) • The results of replication studies of previous findings should be reported even if the results are not significant • Testing for differences in underlying genetic heterogeneities in case and control groups should be performed and reported • Appropriate correction for multiple comparisons across all statistical tests examined should be reported. Comparison to genome-wide thresholds should be described. Similarly, for bayesian approaches, the choice of prior probabilities should be described
Source: NCI-NHGRI Working Group on Replication in Association Studies. NATURE, Vol. 447, 7 June 2007.

5.3 Cumulative genetic difference models

Cumulative genetic difference models are models in which the association between each SNP individually and the phenotype or population is stated for each candidate marker one by one. The most significant markers following some of the criteria described below are added to a list of "n best candidates". For the particular case

of disease phenotypes, once the SNPs found in that list are replicated and confirmed by independent studies, the risk (probability of being a case) of an individual whose phenotype is unknown is assessed using some increasing function over the number of risk alleles found in his/her genome.

5.3.1 Cochran-Armitage test

We include this measure for the sake of completeness even if it is not used in our study. It is a simple extension of the χ^2 test for trends that is superseded by the measures described in the rest of this section.

A simple way to assess the association between a categorical two state variable and a categorical two state phenotype $(case, control)$ is a χ^2 test. Since the categories for a SNP are three RR, RA, AR (missing data can be case-wise excluded in one variable studies) the corresponding test should consider the trend. The equivalent to a χ^2 test for 3 categories considering the trend is the Cochran-Armitage test for trends.

For two phenotype categories, the trend test statistic is:

$$T = \sum_{i=1}^{k} t_i (N_{1i} R_2 - N_{2i} R_1) \tag{5.32}$$

where $k = 3$, N_{ij} is the number of individuals $(i = 1\ cases, i = 2\ controls)$ with allele $(j = 1\ RR,\ j = 2\ RA,\ j = 3\ AA)$, t_i are weights described below $R_i = \sum_{j \epsilon \{1..3\}} N_{ij}$ are row totals, $C_j = \sum_{i \epsilon \{1,2\}} N_{ij}$ are column totals and $N = \sum_{j \epsilon \{1..3\}} C_j = \sum_{i \epsilon \{1,2\}} R_i$ is the total. Under the null hypothesis, this statistic is distributed:

$$\frac{T}{\sqrt{Var\,(T)}} \sim N(0,1) \tag{5.33}$$

being:

$$Var(T) = \frac{R_1 R_2}{N} \left(\sum_{i=1}^{k} t_i^2 C_i (N - C_i) - 2 \cdot \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} t_i t_j C_i C_j \right) \tag{5.34}$$

The choice for t_i is done as follows based on the inheritance model to be tested. In order to test a dominant model where allele R is dominant over allele A, the choice $t = (1, 1, 0)$ is locally optimal. In order to test a recessive model where allele R is recessive to allele A, the optimal choice is $t = (0, 0, 1)$. To test whether alleles R (or A) have additive effects, the choice $t = (0, 1, 2)$ is locally optimal. For complex diseases, the underlying genetic model is often unknown but most often association studies assume the additive version of the test based on Bateson & Yule initial model from 1902 to explain genetics underlying complex and quantitative traits.

5.3.2 Allele frequency

Another way to assess the difference is using the allele frequency in both populations. The frequency of Allele A in a SNP with possible alleles A and T and whose genotypes AA, AT and TT have been observed n_{AA}, n_{AT} and n_{TT} times, is defined as:

$$p_A = frequency\ of\ A = \frac{n_{AA} + \frac{1}{2}n_{AT}}{n_{AA} + n_{AT} + n_{TT}} \tag{5.35}$$

The simplest idea is to measure absolute allele frequency difference between the two populations. This idea has some limitations that have been improved by two other statistics. These statistics are generalized to k subpopulations, i.e., an k-category phenotype: the Weir & Cockerham θ_{ST} genetic distance [107], and the informativeness of assignment index I_n [108]. The author derived a program named SNPInfostats from his GEM library described below in 6.2 to compute these statistics for an entire set of SNPs returning a list of n-highest values to asses North African influences and potential bias in case-control association studies in the Spanish population in [5].

5.3.3 The Weir & Cockerham θ_{ST} genetic distance

Is a variance based method for estimation of F-statistics, which measure the correlation of pairs of alleles between individuals within a sub-population. The null hypothesis assumes the absence of population structure. Alternatively, alleles found within a subpopulation may be found more often together than expected from the entire population.

$$\theta_{ST} = \frac{\sum_i \sum_u \sigma_B^2}{\sum_i \sum_u \sigma_T^2} \tag{5.36}$$

is defined over all alleles i and all loci u. The total variance (in allele frequency)

$$\sigma_T^2 = \sigma_B^2 + \sigma_W^2 + \sigma_I^2 \tag{5.37}$$

is the sum of σ_B^2 between subpopulation variance, σ_W^2 between individuals within subpopulation variance, and σ_I^2 between gametes (each half of the genotype) within individuals variance.

When sample sizes are large enough, this statistic is approached by:

$$\theta_{ST} = \frac{\frac{\sum_j n_j (p_j - \bar{p})^2}{(k-1)\bar{n}}}{\bar{p}(1-\bar{p})} \tag{5.38}$$

where p_j and n_j are the observed allele frequency and sample size of the j-th population, and $\bar{p}$ and $\bar{n}$ are the average allele frequency and sample size for the entire population.

The distribution of this statistic is not used to compute a p-value. The statistic defines an order over the SNPs and the n-highest values are taken in consideration. If the former was required, an empirical distribution could be computed using Monte-Carlo methods.

5.3.4 Informativeness of assignment index I_n

This index has an information theoretic motivation. It is a measure of Kullback-Leibler information divergence [79] under certain number of assumptions, which are not tested.

$$I_n = \sum_{j=1}^{n}(-p_j \log p_j + \sum_{i=1}^{k} \frac{p_{ij}}{\mathrm{k}} \log\ p_{ij}) \tag{5.39}$$

where k is the number of populations $i\epsilon\{1..k\}$, $n = 2$ is the number of alleles at a given locus l. The relative frequency for allele j of locus l in population i is $p_{ij}^{(l)}$ and $p_j^{(l)} = \sum_{i=1}^{k} \frac{p_{ij}^{(l)}}{k}$

[108] states: *"When the minimum description length principle* [109] *is used, up to a constant, In equals the expected reduction, upon observation of J, in the length of the optimal coding of the random variable Q. It gives the average (taken across populations) Kullback-Leibler information for distinguishing population-specific allele frequency distributions from the distribution for the "average population." For k = 2, I_n is similar to a previously proposed statistic based on Kullback-Leibler information."*

As in the case of the allele frequency difference and Weir & Cockerham θ_{ST} genetic distance, no theoretical p-value is computed, a list of n-highest values is returned instead.

5.4 Ancestry assignation in genetic studies.

Teasing apart population sources of human samples is extended in most human studies, be it focused on medicine or forensics. The inference of individual **ancestry** (commonly termed "global ancestry") is very useful in some grounds such as for the improvement of the understanding of complex diseases [110-112] and pharmacological responses [113, 114], as well as in the forensic identification of sample donors [115]. In this context, ancestry of an individual is taken to mean the inherited make up resulting from the **genetic influences of an individual's biological ancestors**. Self-declared racial or ethnic categories have been widely used for ancestry assignation in the biomedical literature. However, racial and ethnic categories are proxies for a wide range of factors, potentially genetic and non-genetic. Therefore, while such labels are important for testing, diagnosis and treatment of human diseases and other biomedical contexts [116], it is widely recognized that self-declared ancestry is error prone [117] and is not an adequate descriptor of the biogeographical ancestry of an individual [118]. This issue has been shown to be extremely important in the context of complex diseases, where ancestry ascertainment assists in the interpretation of results from the most heavily applied designs, i.e. case-control association studies, since unrecognized/unconsidered population subdivisions can severely bias association results if not appropriately adjusted for. Thus, identifying a panel of polymorphims capable of recapitulating individual ancestries is therefore important for the medical sciences. In particular, such panel would offer a cost-efficient tool for independent targeted association studies on particular genes or genomic regions (termed replication studies), in order to bolster the support of initial findings and to be able to examine the generalizability of findings to other populations while maintaining standards of the field (see Table 5.11) even when limited budget is available. [119, 120] constitute two fabulous examples of the issues derived from the presence of population subdivisions (commonly known as **population stratification**) in association studies of complex diseases.

Broad-scale genetic analyses among worldwide populations have shown that about 87-96% of human genetic variation is attributable to differences among individuals, while only 4-13% is due to differences between populations [121]. Congruently, genome wide data allows recapitulating groups of individuals that match closely with groups defined by self-identified race or continental ancestry [122]. Therefore, genetic data can reveal an individual's full ancestry, but only to continental and sub-continental levels [123].

Several recent studies have focused on identifying small panels of SNPs showing large allele frequency differences between populations to serve for ancestry inference, termed **ancestry-informative markers**

(AIMs), to allow recapitulating recent population origins of an individual's genome. However, as markers in sexual chromosomes or in genomic regions where no crossover occurs can yield conflicting results [124], particularly for populations that have experienced sex-mediated genetic influences [125], these identifications have focused on autosomal genetic markers. In these studies, AIM sets have been selected based on individual marker tests after a careful removal of the underlying correlation (i.e. linkage disequilibrium) is made, so that the resultant AIM sets are composed of SNPs that are far apart in the genome [126]. Typically, these studies have prioritized the SNPs to be picked up for ancestry assignment based on ancestry information content measures such as the ones described in Chapter 5, namely I_n, θ_{ST} and Δ [108, 127], or Principal Component Analysis (PCA) eigenvalues [128, 129]. In addition, these methods have relied on the ancestry information recapitulated by single markers, which is not necessarily the same as the one extracted through multipoint (combined) analyses [130].

PCA has been successfully applied to describing sets of AIMs, by retaining the top hundreds that can accurately reconstruct population differences, that are used to control population stratification in association studies [129]. However, such studies have only been attempted to derive AIMs for European and American populations [126, 129, 131-134] so that an extension of such studies to bring up AIMs for other continents continues to be a necessity. In particular, AIMs for populations whose recent ancestors are not represented by any of the currently used reference samples (e.g. HapMap) are lacking. In fact, this was the main problem we faced when aimed to apportion ancestry in an admixed population from Europeans and North Africans [5]. In this context, providing AIM sets for worldwide use is of great value for ancestry inference across human population groups around the world.

In an attempt to describe an efficient and small set of AIMs for worldwide population sets different authors have put forward less than 15 carefully ascertained AIMs for accurate assignation to the 3-4 out of the 5 major continental population subdivisions that have been evidenced across studies [127, 128]. [135] also obtained a set of AIMs for diverse global populations by employing PCA repeatedly through a population-based decision tree to allow optimal individual assignations to populations, supporting that only 50 and 350 carefully selected SNPs would allow precise individual classification to 5 and seven continental regions, respectively, using a consensus nearest-neighbor algorithm with cross validation. Noteworthy, the authors found that classification models failed mostly when aiming to classify individuals of European descent (65.4% accuracy with 300 SNPs).

5.5 The n-factor GEP

In section 5.3 we described two simple (5.3.1 and 5.3.2) and two more advanced (5.3.3 and 5.3.4) measures of association between a SNP and a categorical phenotype. The simplest idea would be to evaluate association for each SNP individually and make a list of the most associated SNPs. This was done in the first implementation of our GEM system in [5]. Such models work to some extent, but are far from being the best n-factor models found, as we will see in the experimental results in 5.8.

5.5.1 Classification models with n SNPs

From a biological point of view, n-factor models can aim at more ambitious targets like inferring the biological pathway [136], detecting haplotype [137] or structuring traits in a network [138]. This is beyond the scope of our study since it requires additional (biological) knowledge and we intend to keep our implementation of MCTS to the n-factor GEP as simple as possible.

Interest in discovering combinations of SNPs with combined high association, rather than just combinations of (individually) highly associated SNPs, has only being paid attention recently [39] and it is a hard and not much explored computation field today. Besides the caveats already mentioned in 5.2, this problem is a case of the well known statistical model selection problem [139-143]. Among the mathematical reasons explaining why the "best model" surpasses the "model of the best", correlation and information redundancy between the variables should be noted. In biological terms: linkage disequilibrium and high complexity of the pathways. This problem has a very large spectrum of applications including medical science, forensics, population genetics, agriculture, etc.

In this dissertation, we define the ***n*-factor GEP** as the problem of finding set of n SNPs that best classifies a population based on a categorical phenotype using the following algorithm.

5.5.2 The classification method

A population from which we have genotype and phenotype files as described in page 110 is divided in two sets: the learning set P_t^L and the test set P_t^T. Note that phenotype is used here only to assign the group membership of each sample and can be generally referred to as the categories or populations. This is done n_{CD}

(number of cross-validation draws) times $t \in \{1..n_{CD}\}$ to avoid the classification depending on specific cross-validation draws. The whole classification procedure is repeated each time and the result is averaged. The size of the test set $|P_t^T|$ is a configurable parameter.

Let $G = \{RR, RA, AA\}$ be the set of possible genotypes for each SNP $i \in \{1,..N\}$, N is the number of SNPs in the model and $\widetilde{G_{.i}}$ is a probability distribution over G. For each subpopulation k $\in \{1,..K\}$ where K is the number of categories in the phenotype, $\widetilde{G_k}$ is a vector formed by the $\widetilde{G_{ki}}$ distributions learned ignoring missing data in the population P_t^L at each step $t \in \{1..n_{CD}\}$.

Let X be an individual of the test set P_t^T whose population is unknown (for the sake of classification) and whose genotype at locus i is $X_i \in \{RR, RA, AA\}$. We define the function:

$$log\left(X, \widetilde{G_k}\right) = \sum_{i=1}^{N} \log(prob\{"X_i \; is \; drawn \; from \; \widetilde{G_{ki}}"\}) \qquad (5.40)$$

Since, under the hypothesis of statistical independence between the SNPs, the probability that X is drawn at random from the distribution $\widetilde{G_k}$ is the product of the probabilities that each of his SNPs i is drawn from each distribution $\widetilde{G_{ki}}$, $log\left(X, \widetilde{G_L}\right)$ is the logarithm of the probability of "X is drawn at random from the learned distribution of the subpopulation k" under the mentioned hypothesis. Abusing language, we shorten the phrase to "X is a k".

To keep our classification method simple and statistically sound, we based it on the likelihood ratio. Since the categorical phenotype has more than 2 categories, we used the likelihood ratio test for composite hypotheses [144]. It is a generalization of the widely used likelihood ratio test created by Pearson in 1933 [145] and has the same value when $K = 2$.

We compute the $log\left(X, \widetilde{G_k}\right)$ for each $k \in \{1,..K\}$ and note the categories $k^{(1)}$ and $k^{(2)}$ corresponding to the highest and second highest values of $log\left(X, \widetilde{G_k}\right)$.

For each individual $j \in [1..|P_t^T|]$ we note the category it most likely belongs $k_j^{(1)}$ and compute

$$\Lambda_j = log\left(X, \widetilde{G_{k_j^{(1)}}}\right) - log\left(X, \widetilde{G_{k_j^{(2)}}}\right) \qquad (5.41)$$

Note that Λ_j is a measure of the confidence we have in the classification $k_j^{(1)}$ for individual j. Λ_j is a log-likelihood ratio. Since the denominator of the likelihood ratio is the likelihood of the alternative hypothesis, it may not seem obvious why we replaced the alternative hypothesis of a composite hypothesis (which should be the complementary of the null hypothesis "X is a $k^{(1)}$") by the second-most likely hypothesis "X is a $k^{(2)}$". We did so after experimenting with different weights for the second and all the other hypotheses. It turned out that the classification obtains better results from discriminating between the most-likely and the second-best, rather than from the "most likely being right" vs. the "most likely being wrong".

The evaluation of each model is simple. To avoid having to compensate between models classifying different numbers of individuals, all models classify exactly 95% of the individuals in P_t^T. From the total number $|P_t^T|$ of individuals the 5% with lowest Λ_j are discarded. The remaining 95% are classified, without having used their known phenotypes so far, the classified phenotypes are compared with the real phenotypes. The number of well classified individuals averaged over n_{CD} complete repetitions of the classification method are used as the value of a model:

$$V(M) = \frac{1}{n_{CD}} \sum_{t=1}^{n_{CD}} \sum_{j=1}^{|P_t^T|} \delta_{k_j^{(1)}, k_j^{(Ph)}} \cdot I_{\Lambda_j \geq \Lambda_t^{c5\%}} \tag{5.42}$$

where $k_j^{(Ph)}$ is the real phenotype category of the individual j, δ is Kronecker's delta, I is the indicator function and $\Lambda_t^{c5\%}$ is 5th percentile of distribution of the log-likelihood ratio Λ_j at each cross-validation draw t.

In short, the n-factor GEP is the problem of finding the model M of n SNPs maximizing $V(M)$ on a given population P defined by its genotype and phenotype files.

5.6 The MCTS implementation of the n-factor GEP

We implemented MCTS to search the space of n-SNP combinations among hundreds of thousands of candidates. Each tree node represents a model. Each node is created by expanding its parent adding one SNP to the parent model starting from the void root node.

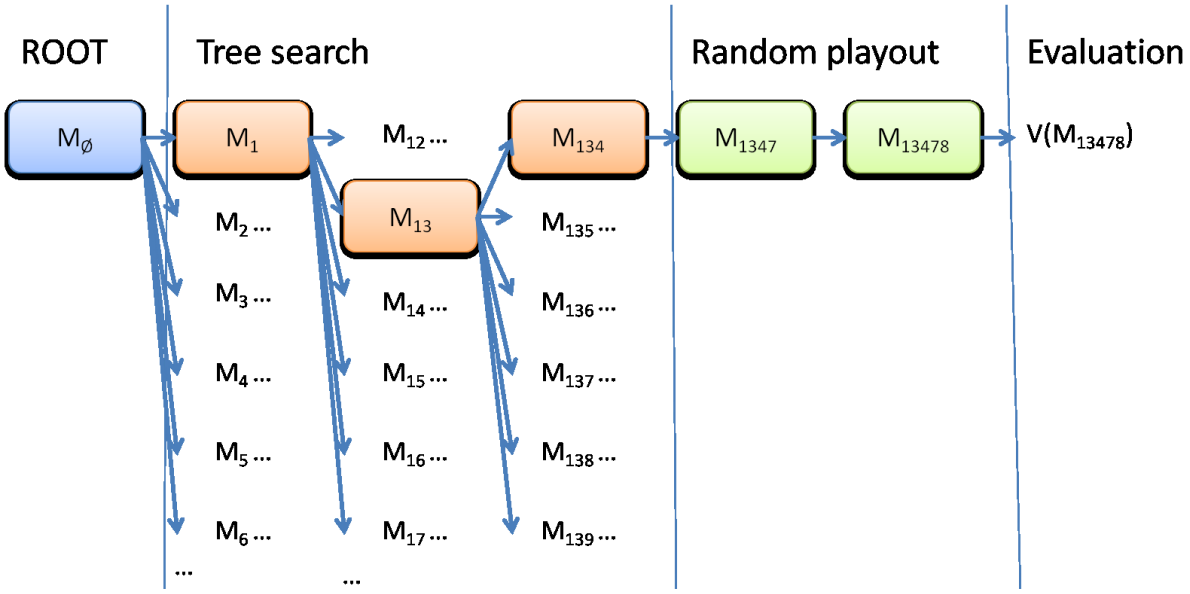

Figure 5.3. Example of an MCTS run evaluating a 5 SNP model with only 9 candidates.

The initial list containing possibly a couple of million SNPs is filtered and evaluated on $V_i^{(p)} = w_1 \cdot I_n + w_2 \cdot \theta_{ST} + w_3 \cdot \Delta$ a weighted combination of I_n as defined in equation 5.39, θ_{ST} as defined in equation 5.38 and a generalized measure of allele frequency difference Δ.

$$\Delta = \frac{1}{n_{kk}} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \left| p_A^i - p_A^j \right| \tag{5.43}$$

where p_A^i is the allele frequency as defined in equation 5.35 for the subpopulation i and n_{kk} is the number of different 2-element combinations of k subpopulations. This list L_{ap} of approximately 50,000 SNPs above some threshold criteria is sorted by $V_i^{(p)}$ and the candidate SNPs are ranked. This rank is used as a priori information for MCTS and also for the GrQS method described below in 5.7.2. The weights (w_1, w_2, w_3) were adjusted by tuning the GrQS algorithm before MCTS was implemented and used without any further tuning in the MCTS algorithm.

Figure **5.3** shows an example of an MCTS run for $n = 5$. The models are composed by introducing the SNPs defined by their rank in the a priori table (1 to 9 in the example). E.g., M_1 is a 1 SNP model using the highest ranked a priori SNP, M_{13} is a 2 SNP model adding the 3rd ranked SNP to the previous one, etc. The search starts considering the first model which is the child of root maximizing equation 5.44. Let's assume it is

M_1. After that, the same applies to the children of M_1 and M_{13} is reached in the example. In the next step M_{134} is reached and, since it is a leaf and the current model has only 3 SNP instead of 5, two more SNPs not already in the model are added to it as a "random playout". The final model is always an n-SNP model and is evaluated as described in 5.5.2. Its value is propagated in the tree upwards in order: $M_{134}, M_{13}, M_1, root$.

Each node i keeps the following information:

- $V_i^{(p)}$ is the a priori value obtained by weighting I_n, θ_{ST} and Δ as described above for the SNP which is added to the parent model at the node.
- n_i is the number of times the node has been visited in the search.
- $\Sigma\big(V(M_i.)\big)$ is the sum of all the evaluations of models built by adding SNPs to this model in the previous search runs.
- The necessary tree structure (pointer to its parent, first child and next sibling) and the necessary model information (IDs of SNPs included).

The path down the tree is done finding at each step the child i maximizing the following V_i

$$V_i = \frac{\Sigma\big(V(M_i.)\big)+V_i^{(p)}}{n_i+1} + K \cdot \sqrt{\frac{\log(n_{p(i)}+1)}{n_i+1}} \qquad (5.44)$$

where K is the constant used in the simplest form of UCT. Also as in the simplest version of UCT, $n_{p(i)}$ is the number of visits of the parent node of i.

When a leaf node is reached and the number of SNPs is below n, a random selection of SNP is added by a non-uniform random draw. The candidate SNPs are drawn using a Bradley-Terry model [146] with a weight system that uses the rank in the a priori table L_{ap} in inverse order $w_i = n_{ap} - r_i + 1$ where n_{ap} is the size of the a priori table and r_i is the rank of SNP i. E.g., in a table with 50000 SNPs the 6th ranked will have a weight 49995 and the last one a weight of 1. Only the SNPs with ranks above the last already in the model are considered since, to avoid repetitions, models are always considered with their SNPs in increasing rank in the candidate table L_{ap} order.

When a leaf node is reached, the number of SNPs is below n and the leaf already has previous visits, the leaf is expanded. Expansion is just adding all children to the tree with zero visits and their a priori value copied

from the candidate table L_{ap}. The first candidate added is the successor of the last one already in the model to avoid wasting computer resources by inserting the same model more than once in the tree.

When a final n-SNP model is reached, the model is added to L_{best}, a list of best models found so far, when its value is above that of the worst model in the list.

When the final n-SNP was obtained entirely in the tree (without a random playout) its value is penalized (by a small value like the 2%) before propagating upwards to the tree. This is necessary to avoid the algorithm deterministically repeating the same path when the best model is found in the tree, wasting CPU time since it ends further exploration. Note that unlike the playout, the tree search is deterministic. The penalization favors the exploration of all siblings in the tree whose value is close to that of the nodes in the repeated path and will find and alternative promising path to be explored in a natural way. It was tuned experimentally starting with small values and increasing the value until path repetition was avoided fast enough in preliminary tests.

Tree update is done by increasing the number of visits and adding the value of the final model to the $\Sigma(V(M_{i.}))$ field of the nodes in the path visited. Update is done starting with the leaf and visiting the parent of each node until root is updated.

When the assigned computation time is over, the list of best models L_{best} is returned.

5.7 Other methods used in the experiments

This section describes the methods used in the experimental section to compare with MCTS. There is no established regression model to predict phenotype that could help comparing our results with those of other authors. Neither is there a public data set intended for that purpose and no algorithm is a *de facto* standard since the field is still in its early days. For that reason, we used two simple methods, one is the top list of n SNPs which is a mandatory reference point and provides evidence supporting the claims made in 5.5.1 that the "best model" in not the "model of best" and the other is a highly optimized method used to select categorical factors in statistical regression with suitable properties for the experiments.

5.7.1 TopN: Best of n-best based on genetic difference models

TopN returns 3 models which are the models formed by n-SNP with the highest values of:

- List 1. I_n as defined in equation 5.39
- List 2. Δ as defined in equation 5.43
- List 3. θ_{ST} as defined in equation 5.38

This algorithm is just a reference point and does not benefit from additional CPU allocation. The statistics in the candidate list L_{ap} described in 5.6 is computed in advance. This algorithm just evaluates the three models and returns the result for comparison with the other two algorithms. This is done almost immediately with no substantial part of the allocated CPU time used.

5.7.2 GrQS: Greedy queued search applied to the n-factor GEP

GrQS a greedy incomplete brute force search using an infinite priority queue. The priority queue has a physical size in excess of the number of models that could be evaluated in the allocated CPU time. For that reason, when the queue is filled, models with a priority below that of the worst model in the queue are discarded. This behavior is logically equivalent to using an infinite priority queue since the discarded models would not have been evaluated in time anyway. The implementation of the priority queue is very efficient due to the use of AA trees [147]. This algorithm uses the same candidate list L_{ap} described in 5.6. Just like the MCTS algorithm, the algorithm starts with an empty model and adds SNPs to it one by one in a similar way as the tree expansion works in MCTS. Instead of evaluating the models, their value is estimated which is much faster. The estimated value of a model is $\widehat{\mathrm{V}}_{\mathrm{i}}$:

$$\widehat{\mathrm{V}}_{\mathrm{i}} = \mathrm{V}_{\mathrm{p(i)}} + K_{n(p(i))} \cdot V_i^{(p)} \tag{5.45}$$

where $\mathrm{V}_{\mathrm{p(i)}}$ is the (evaluated) value of the parent, $K_{n(p(i))}$ is a constant (a slope) that represents the expected increase in value due to adding one SNP to the model and is therefore a function of the number of SNPs in the parent model $n\big(p(i)\big)$ tuned by averaging many n-SNP models in preliminary tests. As before, $V_i^{(p)}$ is the a priori value breaking the ties within the children of the same parent anything else being equal in that case. This estimated value is the priority with which the models wait in the queue.

The algorithm works as follows: The queue is initialized with all the children of the void model (the 1 SNP models) estimated by equation 5.45. At each step of the algorithm, the model with the highest priority is read out of the queue and evaluated using the classification procedure described in 5.5.2. If the model has less than n SNPs, all its children are estimated by the equation 5.45 and pushed to the queue with their corresponding priorities. If the model has n SNPs, the model is added to L_{best}, a list of best models found so far, when its value is above that of the worst model in the list.

Just like in MCTS, when the assigned computation time is over, the list of best models L_{best} is returned. Unlike the MCTS algorithm, this algorithm also evaluates models with less than n SNPs. A poor choice of the constants $K_{n(p(i))}$ (too optimistic estimation does not fit approximately the real values obtained from evaluation) could result in too much exploration (rather than exploitation) and the algorithm not reaching n-SNP models fast enough. On the opposite side, too pessimistic estimation could result in a model immediately reaching n SNPs and the algorithm changing only the final SNPs in the model without exploring alternatives to the first SNPs in it. Besides these important caveats, the algorithm is not hard to tune. Because of the hierarchical structure of the models, analyzing the pace at which models representing alternatives for the first SNP reach the end of the queue gives excellent feedback to tune the algorithm. For the problems in the experiments, the same set of values could be used across the three problems resulting in suitable balance between exploration and exploitation. These values were adjusted to match the allocated CPU time (more exploration for larger times). Automatic tuning could be a future improvement to consider.

The need of tuning is also an argument in favor of MCTS, whose only tunable constant K is far easier to interpret and the algorithm is robust enough not to need any tuning if the evaluation function had been scaled to a fixed range, e.g., $[0, 1]$.

In favor of GrQS, this algorithm converges to an exhaustive brute force search if the problem had been small enough for this to be feasible. Limiting the number of candidate SNPs the algorithm would efficiently find the best model without ever repeating an evaluation. MCTS is known to converge to perfect play when applied in a minimax search to a $\{win, loss\}$ output, but in optimization problems, the algorithm could cycle even with the penalization described in 5.6. The number of models computed more than once is also very high, except when the search space is so big that the explored part is scarce even for the highest CPU times like in the experiments. This issue is further discussed in 7.2.6 and references to MCTS applied to optimization problems can be found in 7.1.2.

5.8 Experimental results

We conducted a series of experiments to compare the three algorithms described above: MCTS (see 5.6), GrQS (see 5.7.2) and TopN (see 5.7.1). We used a dataset of 1043 human individuals from the entire planet from the public genome wide dataset Human Genome Diversity Project [40] we created three different problems, World 7, World 5 and World 3 dividing the global human population in 7, 5 and 3 categories respectively. We used standardized categories described in [41] for 5 and 7 categories and removed the less represented categories: Americas (108 individuals in all) and Oceania (36 individuals) for the World 3 problem.

Each of the 3 problems (World3, World 5 and World 7) was run by each of the 3 algorithms (MCTS, GrQS and TopN) with 3 different model sizes (4 SNPs, 8 SNPs and 12 SNPs) and with 3 different CPU allocation times (10 minutes, 40 minutes and 160 minutes). Note that the TopN algorithm does not scale with CPU allocation and always returns the same results without using the allocated time.

The number of individuals in the test set $|P_t^T|$ was 400 for the World 7 and World 5 problems and 300 for World 3. The number of individuals used for learning $|P_t^L|$ were 643 in the World 5 and World 7 problems and 599 in World 3. As mentioned, the 5% of individuals with lower log-likelihood ratio Λ_j as explained in 5.5.2 were not classified. I.e., the value of each model is a number in $\{0,..380\}$ of well classified individuals for World 5 and World 7 and $\{0,..285\}$ for World 3.

Each time the model is evaluated, the classification procedure is repeated $n_{CD} = 5$ times on different randomly drawn cross-validation sets. The cross validation populations are pre-computed. Therefore, each time the same model is evaluated, it returns exactly the same value: the average of 5 cross-validation draws on the pre-computed population.

Table 5.12. Human Genome Diversity Project populations in our problems.

	Pop. name	Origin	Count	In World 7	In World 5	In World 3
1	Bantu Kenya	Kenya	12	Africa	Africa	Africa
2	Bantu South Africa	South Africa	8	Africa	Africa	Africa
3	Biaka Pygmy	Central African Republic	32	Africa	Africa	Africa
4	Mandenka	Senegal	24	Africa	Africa	Africa
5	Mbuti Pygmy	Democratic Republic of Congo	15	Africa	Africa	Africa
6	San	Namibia	6	Africa	Africa	Africa
7	Yoruba	Nigeria	24	Africa	Africa	Africa
8	Colombian	Colombia	13	Americas	Americas	---
9	Karitiana	Brazil	24	Americas	Americas	---
10	Maya	Mexico	25	Americas	Americas	---
11	Pima	Mexico	25	Americas	Americas	---
12	Surui	Brazil	21	Americas	Americas	---
13	Cambodian	Cambodia	11	E Asia	E Asia	E Asia
14	Dai	China	10	E Asia	E Asia	E Asia
15	Daur	China	9	E Asia	E Asia	E Asia
16	Han	China	34	E Asia	E Asia	E Asia
17	Han N China	China	10	E Asia	E Asia	E Asia
18	Hezhen	China	9	E Asia	E Asia	E Asia
19	Japanese	Japan	29	E Asia	E Asia	E Asia
20	Lahu	China	10	E Asia	E Asia	E Asia
21	Miao	China	10	E Asia	E Asia	E Asia
22	Mongola	China	10	E Asia	E Asia	E Asia
23	Naxi	China	9	E Asia	E Asia	E Asia
24	Oroqen	China	10	E Asia	E Asia	E Asia
25	She	China	10	E Asia	E Asia	E Asia
26	Tu	China	10	E Asia	E Asia	E Asia
27	Tujia	China	10	E Asia	E Asia	E Asia
28	Xibo	China	9	E Asia	E Asia	E Asia
29	Yakut	Siberia	25	E Asia	E Asia	E Asia
30	Yi	China	10	E Asia	E Asia	E Asia
31	Balochi	Pakistan	25	CS Asia	Eurasia	Eurasia
32	Brahui	Pakistan	25	CS Asia	Eurasia	Eurasia
33	Burusho	Pakistan	25	CS Asia	Eurasia	Eurasia
34	Hazara	Pakistan	24	CS Asia	Eurasia	Eurasia
35	Kalash	Pakistan	25	CS Asia	Eurasia	Eurasia
36	Makrani	Pakistan	25	CS Asia	Eurasia	Eurasia
37	Pathan	Pakistan	23	CS Asia	Eurasia	Eurasia
38	Sindhi	Pakistan	25	CS Asia	Eurasia	Eurasia
39	Uygur	China	10	CS Asia	Eurasia	Eurasia
40	Adygei	Russia Caucasus	17	Europe	Eurasia	Eurasia
41	Basque	France	24	Europe	Eurasia	Eurasia
42	French	France	29	Europe	Eurasia	Eurasia
43	Italian	Italy (Bergamo)	13	Europe	Eurasia	Eurasia
44	Orcadian	Orkney Islands	16	Europe	Eurasia	Eurasia
45	Russian	Russia	25	Europe	Eurasia	Eurasia
46	Sardinian	Italy	28	Europe	Eurasia	Eurasia
47	Tuscan	Italy	8	Europe	Eurasia	Eurasia
48	Bedouin	Israel (Negev)	48	Middle East	Eurasia	Eurasia
49	Druze	Israel (Carmel)	47	Middle East	Eurasia	Eurasia
50	Mozabite	Algeria (Mzab)	30	Middle East	Eurasia	Eurasia
51	Palestinian	Israel (Central)	51	Middle East	Eurasia	Eurasia
52	Melanesian	Bougainville	19	Oceania	Oceania	---
53	Papuan	New Guinea	17	Oceania	Oceania	---

Note: The two major human categorizations used in our problems: World 7 and World 5 were taken from Noah A. Rosenberg, *"Standardized subsets of the HGDP-CEPH Human Genome Diversity Cell Line Panel, accounting for atypical and duplicated samples and pairs of close relatives"*. World 3 was created removing the two categories with smaller population counts in the World 5 dataset.

Table 5.12 shows the categories for each of the problems in colors grouping the 53 populations in the HGDP dataset in standardized classes.

Some algorithm-specific results and settings used in the experiments were:

- **MCTS specific**: The UCT constant K in equation 5.44 was optimized at $K = 40$.
- **TopN specific**: From the 3 lists, the best classifier across all experiments 3 (problems) × 3 (number of SNPs) × 1 (all allocation times return the same models): The list based on maximum values of I_n was the best 5 times of 9. The lists based on θ_{ST} and Δ were both the best twice in 9. The list based on θ_{ST} was the worst (4 of 9), I_n was the worst (3 of 9) and Δ was the worst (2 of 9). Also, when weighting the influence in the list of candidates L_{ap}, best results were obtained by giving more weight to I_n and less to θ_{ST} with the weight of Δ in between.

5.8.1 Average performance of the best model

Focusing on the algorithms, rather than the individual problems, table 5.13 shows the results obtained by each algorithm across the 3 problems. Results are in percentage of 380 for the problems World 5 and Word 7 and 285 for Word 3 and the displayed value is the average of the 3. The percentage can be stated as "percentage of well classified individuals in the most confident 95% of the classified population".

Table 5.13. Average performance of the best model

	n SNP = 4	n SNP = 8	n SNP = 12
Time 600"	MCTS 88.1±1.1 (p=0.0002) GrQS 85.8±1.8 (p<0.0001) TopN 78.5±2.3	MCTS 93.3±0.5 (p<0.0001) GrQS 89.4±0.8 (p<0.0001) TopN 87.0±1.5	MCTS 94.6±0.6 (p<0.0001) GrQS 92.8±0.6 (p<0.0001) TopN 89.4±1.5
Time 2400"	MCTS 88.1±1.1 (p=0.0332) GrQS 87.2±1.1 (p<0.0001) TopN 78.5±2.3	MCTS 93.3±0.6 (p<0.0001) GrQS 90.0±1.2 (p<0.0001) TopN 87.0±1.5	MCTS 94.8±0.6 (p<0.0001) GrQS 93.0±0.5 (p<0.0001) TopN 89.4±1.5
Time 9600"	MCTS 88.1±1.1 (p=0.7595) GrQS 88.0±0.6 (p<0.0001) TopN 78.5±2.3	MCTS 93.5±0.9 (p<0.0001) GrQS 91.4±0.7 (p<0.0001) TopN 87.0±1.5	MCTS 95.3±0.3 (p<0.0001) GrQS 93.4±0.8 (p<0.0001) TopN 89.4±1.5

Note: Each cell represents the best model averaged across the three problems: World 7, World 5 and World 3. The values represent percentage of well classified individuals ± SD (standard deviation). Algorithms are sorted best first. Each evaluation is done on five independent random draws of the dataset, selecting different learning and training subsets. P-values measure evidence that the algorithm's average performance is above that of the next algorithm, MCTS > GrQS and GrQS > TopN, using a single sided t-test for independent samples. Times represent computing time in seconds for each search: one problem, one number of SNP, one algorithm (GrQS and MCTS).

In all the cases, MCTS was the best algorithm performing better than GrQS and GrQS performed better than TopN for all model sizes and CPU allocations. Additionally, the tables show that this difference is statistically significant with significance above 99% for all models larger than 4 SNPs using a t-test for independent samples.

5.8.2 Best model for each problem

Table 5.14. Best model for each problem

	n SNP = 4	n SNP = 8	n SNP = 12
Problem World 3	MCTS 282.6± 1.7(99.2%)	MCTS 284.6± 0.5(99.9%)	MCTS 284.8± 0.4(99.9%)
	GrQS 281.6± 0.5(98.8%)	GrQS 284.0± 0.7(99.6%)	GrQS 284.2± 0.8(99.7%)
	TopN 272.6± 4.2(95.6%)	TopN 273.6± 4.6(96.0%)	TopN 273.0± 5.1(95.8%)
Problem World 5	GrQS 359.8± 3.6(94.7%)	MCTS 376.0± 1.7(98.9%)	MCTS 379.0± 0.7(99.7%)
	MCTS 359.4± 4.9(94.6%)	GrQS 373.4± 2.5(98.3%)	GrQS 375.4± 1.7(98.8%)
	TopN 301.4± 9.0(79.3%)	TopN 357.2± 3.9(94.0%)	TopN 360.4± 4.1(94.8%)
Problem World 7	MCTS 268.2± 5.6(70.6%)	MCTS 310.0± 7.9(81.6%)	MCTS 327.4± 2.3(86.2%)
	GrQS 268.2± 2.2(70.6%)	GrQS 289.8± 4.4(76.3%)	GrQS 310.8± 5.8(81.8%)
	TopN 230.6±11.7(60.7%)	TopN 269.6± 7.2(70.9%)	TopN 294.4± 6.4(77.5%)

Note: Each cell represents the best model evaluated on five independent random draws of the dataset, selecting different learning and training subsets. The values represent total number of well classified individuals ± SD (standard deviation). The algorithms classify 285 individuals in the problem World 3 and 380 individuals in the problems World 5 and World 7. Percentages in brackets represent the proportion of correctly classified from the number of classified individuals. Algorithms are sorted best first. All values are obtained from the best model found in the longest search (9600 seconds).

Table 5.14 shows the best model (obtained from the maximum CPU time allocated) for each of the problems. As would be expected from the large performance differences obtained across the problems with more than 4 SNPs, MCTS was the best in each of the problems with more than 4 SNPs and not only on average across the problems. In problem World 5 with 4 SNPs GrQS performed slightly better than MCTS although that difference (359.8± 3.6 vs. 359.4± 4.9) is not statistically significant (p=0.4433) using a t-test for the mean. This was the only case in which GrQS performed better than MCTS.

5.8.3 Average performance of the second and third-best models found

Table 5.15. Average performance of the second and third-best models found

	n SNP = 4	n SNP = 8	n SNP = 12
Time 600"	MCTS 88.1±1.1 88.0±1.2	MCTS 93.2±0.6 93.2±0.5	MCTS 94.4±0.4 94.3±0.6
	GrQS 85.7±1.5 85.5±1.2	GrQS 89.1±1.5 89.1±0.9	GrQS 92.7±0.6 92.7±0.7
	TopN 70.5±1.5 67.8±1.3	TopN 84.3±1.3 77.7±1.4	TopN 88.3±1.1 85.1±1.8
Time 2400"	MCTS 88.1±1.1 88.0±1.0	MCTS 93.3±0.7 93.3±0.6	MCTS 94.8±0.6 94.8±0.5
	GrQS 86.9±1.5 86.6±1.2	GrQS 89.9±1.0 89.8±1.1	GrQS 93.0±0.9 92.9±0.7
	TopN 70.5±1.5 67.8±1.3	TopN 84.3±1.3 77.7±1.4	TopN 88.3±1.1 85.1±1.8
Time 9600"	MCTS 88.1±1.1 88.1±1.1	MCTS 93.4±0.9 93.4±0.9	MCTS 95.3±0.3 95.3±0.3
	GrQS 87.9±0.9 87.8±1.0	GrQS 91.3±0.5 91.2±1.0	GrQS 93.3±0.6 93.3±0.8
	TopN 70.5±1.5 67.8±1.3	TopN 84.3±1.3 77.7±1.4	TopN 88.3±1.1 85.1±1.8

Note: Each cell represents the second and third-best models averaged across the three problems: World 7, World 5 and World 3. The values represent percentage of well classified individuals ± SD (standard deviation) second-best model found followed by third-best model found. Algorithms are sorted by second-best model. Each evaluation is done on five independent random draws of the dataset, selecting different learning and training subsets. Times represent computing time in seconds for each search: one problem, one number of SNP, one algorithm (GrQS and MCTS).

Further insight on how all algorithms consistently succeed in finding more models at (or near) their best performance is shown in table 5.15. In most cases, the 3rd best model found by each algorithm performs better

than the best of the next algorithm. As in the previous cases, the difference is smaller for the models with less SNPs (n SNP = 4).

5.8.4 Worst evaluation in the best model

Table 5.16. Worst evaluation in the best model

	n SNP = 4	n SNP = 8	n SNP = 12
Time 600"	MCTS 86.7	MCTS 92.6	MCTS 93.7
	GrQS 83.4	GrQS 88.1	GrQS 91.9
	TopN 75.4	TopN 85.1	TopN 87.4
Time 2400"	MCTS 86.6	MCTS 92.5	MCTS 94.0
	GrQS 86.0	GrQS 88.8	GrQS 92.2
	TopN 75.4	TopN 85.1	TopN 87.4
Time 9600"	GrQS 87.5	MCTS 92.3	MCTS 94.8
	MCTS 86.6	GrQS 90.4	GrQS 92.4
	TopN 75.4	TopN 85.1	TopN 87.4

Note: Each cell represents the worst evaluation in the five random draws of the dataset averaged across the three problems: World 7, World 5 and World 3. The values represent percentage of well classified individuals in the worst of five evaluations. The algorithms are sorted best first. This is a lower bound for the performance of each algorithm. Times represent computing time in seconds for each search: one problem, one number of SNP, one algorithm (GrQS and MCTS).

Like in the previous case, in order to establish the consistency of the results, we show the results obtained by the worst evaluation (worst of 5) obtained in the cross-validation draws in table 5.16. Results show, especially for the models with more than 4 SNPs, that even a worst-case classification obtained by each algorithm outperforms average classification of the next algorithm providing further evidence in favor of MCTS.

5.8.5 Improvement with increased CPU allocation

Table 5.17. Improvement with CPU allocation before the best model was found

	Time 600"	Time 2400"	Time 9600"
n SNP = 4	MCTS 14.8	MCTS 37.5 (d=0.00)	MCTS 7.7 (d=0.02)
	GrQS 38.7	GrQS 32.9 (d=1.35)	GrQS 24.0 (d=2.19)
n SNP = 8	MCTS 60.9	MCTS 55.4 (d=0.05)	MCTS 33.4 (d=0.20)
	GrQS 70.6	GrQS 43.0 (d=0.61)	GrQS 60.0 (d=2.01)
n SNP = 12	MCTS 89.9	MCTS 67.9 (d=0.25)	MCTS 43.7 (d=0.68)
	GrQS 29.6	GrQS 19.0 (d=0.25)	GrQS 38.2 (d=0.67)

Note: Each cell represents the percentage of CPU time required to find the best model from the total time assigned to each search averaged across the three problems: World 7, World 5 and World 3. The value shown as d is the difference in average performance of the best model achieved by longer computing time as a difference with the base performance (Time = 600 sec).

Table 5.17 gives some insight about the time needed to find each of the best models. A full scalability study would require a large amount of CPU allocation and possibly more problems and datasets. At least for

the times chosen, each quadruplication in allocated time resulted in better performance as shown by $d > 0$ for all results, except when SNP = 4. Some amount of diminishing returns should be considered normal, especially for n SNP = 4 which is clearly scaling less than the other sizes. We do not have enough information to plot a performance vs. time graph. Also, some implementation details concerning memory allocation and estimation of effective width at node expansion could play a major role. Mechanisms to recycle least-recently visited nodes should be considered for a more robust version of the program designed to compute over long periods. The current version just stops node expansion when RAM is exhausted, but that did not happen in the experiments. At this stage, results provide evidence that the problem is hard and benefits from longer (hours) computation at least for the larger problems.

Chapter 6. Methods: Description of the software used in this thesis

6.1 The GoKnot platform

GoKnot is a graphical board interface and playing client entirely written by the main author for over 10 years. The project started in 2002 and version 1.0 was released in February 2003. It started as proprietary software, although when version 2.0 was released in November 2005, the interface with the engine, named **gke** together with a complete gnugo based engine were released as free software. GoKnot had over 30,000 downloads since version 2.0 was released [148].

GoKnot is also an sgf editor and sgf-syntax converter and its board rendering utilities can also be used for generating documentation. All *go* boards included in this dissertation were rendered by GoKnot.

Since then, no more releases have been made public so far. Work on GoKnot has continued without interruption prioritizing implementation of a strong engine and a development environment rather than spending time documenting and supporting work in progress. GoKnot itself is the GUI editor, client and analysis tool. The term GoKnot platform is used to describe all utility programs, libraries, and datasets including the isGO (MCTS) *go* engine (formerly named QYZ).

Table 6.18 describes the main modules giving an idea of the size of this ten year long software project and the amount of work necessary for implementing a strong *go* engine. Especially, if the implementation is focused on new research rather than just on implementing successful ideas. This section also describes some of

the free libraries and datasets and some results on move prediction based on the patterns managed by the HBS board system.

Table 6.18. List of modules in the GoKnot platform

Name	Description	Started (approx. # of lines) Language	License
GoKnot.exe	The GUI, local and internet playing client, analysis tool, game editor and development environment for the engine	2002 (23,317) CP(*)	Prop
Sgfc.exe	SGF syntax checker and converter from old versions	2004 (5,495) C(*)	Free
gkGTP.exe	GTP end for the GUI environment. Connects the active engine in the GUI to other GTP-compatible programs and internet servers.	2007 (1,117) P(*)	Prop
GKE: types library	Types library for writing GKE compatible engines. GKE engines are loaded in RAM without interprocess barriers and support MCTS specific analysis and debugging tools.	2004 (1,103) P(*)	Free
GKE: GTPeng.dll	And example open source GKE engine that links any gtp-enabled engine to GoKnot and GKE tools	2008 (1,736) P(*)	Free
HBS: wrCrJeit.exe	Pascal program writing board urgency pattern management functions in automatically written assembly language	2008 (918) P(*)	Prop
HBS: wrUpNeib.exe	Pascal program writing proximity management patterns in automatically written assembly	2008 (863) P(*)	Prop
HBS: brdTest.exe	Exhaustive board functionality testing program that verifies 100% of the functionality of the assembly language code and other high level board functionality in all modes and sizes	2008 (1,492) P(*)	Prop
HBS: ASM board source	Automatically written board pattern management	2008 (53,855)A(*)	Free
HBS: board source	Board level functionality: stone playing, legality, Zobrist hashing, liberty counts, patterns	2008 (7,348) ACP(*)	Prop
HBS: MCTS fundamentals	MCTS engine functionality: tactical utilities, playout functionality, tree management, time control, etc.	2008 (18,798) CP(*)	Prop
MASTERS: UpdMastr.exe	Utility merging the master games database with folders containing .sgf files filtering by criteria and automatically managing version conversion, etc.	2007 (1,304) P(*)	Prop
MASTERS: masters.go	The database of master games	2007 (data)	Free
UTILS: BldUrgen.exe	Utility for extracting from master games the frequently seen local patterns, learning the Bradley Terry models of urgency and testing the results.	2007 (4,150) AP(*)	Prop
UTILS: BldJoseki.exe	Utility extracting *fuseki*/*joseki* from master's db	2008 (4,749) CP(*)	Prop
UTILS: JosekLrn.exe	Utility using the MCTS engine to compute final territory and assigning predicted territory to *joseki* patterns described in M-eval 2.0	2010 (2,211) P(*)	Prop
UTILS: LearnR8.exe	Utility creating .csv dataset files from positions in the master's db for NMF analysis	2008 (2,142) P(*)	Prop
UTILS: regTest.exe	Utility for testing GKE engines on regression problems stored in Fuego-compatible text files. The output is written in .csv for analysis.	2010 (1,114) C(*)	Prop
UTILS: TwoGke.exe	Utility for playing games between two GKE engines (one of them may be a gtp link to another engine). The output, including time information is stored as .csv files for analysis and .sgf files (the games).	2010 (1,462) C(*)	Prop
UTILS: EdO.exe	Utility for managing GKE engine parameter settings for the experiments.	2010 (206) P(*)	Prop
isGO.dll	The engine. Top level management.	2008 (6,358) ACP(*)	Prop

(*) Programming language: Any combination of: A = Pentium II assembly (32 bits), C = C++, P (Delphi) Pascal

6.1.1 The Hologram Board System (HBS)

The board system is named "hologram" because each cell (board intersection) contains information about its neighbors in the form of a bitmap pattern (or a Zobrist hash of the pattern depending on the board mode). This is similar as the way information is stored in a hologram (which is about storing interference patterns rather than picture elements) and shares the property that each part of a hologram "somehow" contains the whole picture. Of course, this is a redundant way to store information, but it is done for the sake of efficiency when checking the pattern. To make the whole idea work, each time a stone is added or a chain is captured, all possibly interacting cells must have their neighboring patterns updated. This is done by over 50,000 lines of automatically written (and 100% error free) assembly language source code. All variables are kept in CPU registers and neither conditional jumps nor loops are used. Also board limits are not coded as conditionals, since each board cell has its own updating function written for its position relative to the limits of the board.

The board system not only supports keeping patterns up to date, but also checking the hash codes of the patterns in a table of *shapes→urgencies*. The first MCTS engine implemented used this distribution over the legal moves as a playout policy (see 6.1.3) and its speed was comparable to that of other strong engines. Unfortunately, the policy was very weak in terms of playing strength. Finally, the hash values of patterns have been used in our playout policy as described in 4.3.2, making the HBS board libraries, again, an interesting option for *go* engines.

Since implementing all the pattern management in assembly language to make it fast enough to be interesting is a complicated and time consuming task, and this is already implemented since about 2009, we have decided to make the board level assembly language libraries open source software [149] hoping other authors will be interested in implementing a similar board system to support learning playouts as described here. Since they are automatically written, customization can in some cases be done to match other ways to store the board information.

6.1.2 The collection of master level games in 19x19

Thanks to the internet, large collections of high quality games can be found. In our research we use one collection with 55,271 games named Masters 2010, about one third is played by pro players and the others by

high *dan* players on internet servers filtering all kind of lower quality games like fast games and handicap games. The website includes the database for download in a binary format (one game per record) that is documented on the website [150].

Searching patterns in this database can be used for:

- *Fuseki*: Full board opening in 19x19 *go* it is not very useful since the engine gets out of the book very soon, but it is still a good idea because the engine gets some free thinking time and the opening looks better. There is no fixed number of opening moves expected. The game gets very fast into a position that is not found the database. A typical book with about 25 thousand positions will only rarely play the first eight moves and that will probably happen against a program with a book extracted from a similar collection. Against humans, the book will only reach to move four or five at most.
- Urgency of the pattern of stones with a fixed size around the next move. It cannot be used as a playout policy, but it gives interesting information about shape, as urgent points are frequently weaknesses in the shape and it still makes part of the most recent versions of M-eval.
- Obtaining datasets for analyzing the multivariate structure of the qualities described in M-eval (see 3.4). The collection can be used as a large set of positions.
- *Joseki*: We analyzed the board divided in regions from different sizes. These regions are small corners, which are not enclosed by empty intersections, big corners which are, and big sides. In these areas we study common patterns seen in a large amount of games and learn their possible answers. Besides what the answers are, we also learn how frequently each answer is found. This gives us a good idea of possible human continuations to moves in places where known patterns are found, even when the rest of the board is too complicated to judge by simple methods. This provides the engine with *joseki* sequences that will be considered together with the candidate moves and compete with them. Of course, this definition of *joseki* only means "frequently found in that place", *joseki* should also be about optimality which we do not analyze here.

6.1.3 Prediction of master level moves using HBS's local patterns

The board system developed for all MCTS engines in GoKnot was conceived with the idea of using urgencies of frequently seen patterns formed by the stones around all intersections to define a probability distribution over the legal moves. The implementation includes over 50,000 automatically generated lines of

assembly source code for each possible pattern size. Pattern sizes range from 4 neighbors to 40 and they are bitmaps of 4 possible states (empty, own color, opponent color, off board). Functionality includes mirroring, rotation and computing Zobrist hashes. The urgency defined as the number of times the pattern was played divided by the number of times it was seen and later improved using Bradley-Terry models was stored in a database of frequently seen patterns independently of mirroring and rotation. The possible sizes and their layout are described in 4.3.2. The code is highly optimized with all possible loops unrolled, no conditional jumps and all results stored in CPU registers. Also, systematic testing of the entire code was performed checking its functionality against an implementation written in a high level language.

This code allowed an implementation that performed very high prediction rate of played moves while being fast enough to produce a number of simulations per second comparable with those of the stronger programs based on benchmarking open source implementations and also performance reported by the authors of proprietary programs. This performance is between 1000 and 2000 sims/second on a 19x19 board for a single 3GHz core depending on pattern size, database size and other settings. Prediction rate cannot be directly compared with values reported by other authors since we only tried to optimize prediction rate among the positions with a played move found in the database. Prediction rate was never a target by itself, but rather a way to sort the candidate moves for the tree search and to increase their exploration in the playouts. Some prediction rate results in summary are: Using a large database containing 239,502 patterns of 40 neighbors, from a set of 4,968,759 random positions tested from a collection of 55,271 games 1,381,595 positions (27.8%) were followed by a move found in the database of frequent patterns. The database was learned from different positions of the same collection of games and its urgencies were adjusted to maximize the chances of guessing the played move. From those positions, the most urgent pattern represented the played move correctly 40.9% of the time and the second best 11.5% of the times. The first 5 candidates included the correct move over 65% of the times (66.5%). The first 7 moves over 70% (71.6). 15 candidate moves were necessary to reach 80% of inclusion of the played move, 62 to reach 85.0% and 99 to reach 90%. In all, only the 17 best moves found in the database had better chances of being the played move than any legal random move.

Unfortunately, that high prediction rate of strong player's moves did not result in strong play. This idea was a serious limitation for the program strength until it was finally abandoned in 2010. Nevertheless, the board implementation efficiently computes bitmap patterns around the legal moves at a very small computational cost and these patterns are essential in the learning playout policy described below in 2.

6.1.4 isGO: the MCTS engine using the HBS

First named QYZ and renamed to isGO when the original playout policy based on playing moves with a probability distribution learned from human play as described above was replaced by the policies described in chapter 4, starting in 2010 an completed in 2012. isGO is based on the GKE interface which is open software [151]. isGO is proprietary software using the libraries mentioned above in 6.1. It is a competitive engine whose playing strength can be compared with that of the strongest open source engines. A comparison with Fuego in terms of strength can be found in the results of the experiments on playouts in chapter 4.

6.2 The GEM library

All the experiments in chapter 5 were done using C++ software written by the main author. This program is a collection of libraries for bioinformatics jointly named GEM [152]: The Gene Etiology Miner. GEM was originated from different projects written by the author, the statistics described in 5.3 were used for the first time in [5] and also taken from other commercial products developed by the author like the fundamentals of the algorithm GrQS (see 5.7.2) which was originally with a different classification method.

The GEM library is still under development. It can do everything implemented in chapter 5 and many of the features in the specification list for the next version, including:

- Read genotype/phenotype files from in variety of formats including support for many commercial DNA chips from major manufacturers, including: Affymetrix, Illumina and Perlegen.
- Include information about genes, linking SNPs with the gene database.
- Automatically provide information to the user about Genome Wide Association Studies mentioning the SNPs used in the models. That information is regularly updated from the database at the National Human Genome Research Institute (NHGRI) [153] and GEM-based software can keep its information synchronized with a centralized server using SOAP technology.
- Provide information (or implement it in the search) about the (predicted) functionality of the SNP. These predictions are updated from two different servers SIFT [154] located at Craig Venter Institute [155] and PolyPhen [156-158] located at Harvard University [159]. Like in the previous case, GEM-based software can be synchronized with a centralized server using SOAP technology.
- Download, update, configure and interact with GEM based programs using any Ajax enabled web browser.

- Implement n factor GEP algorithms as plug-ins to add a new algorithm without reinstalling all the system.
- Distribute CPU workload on different machines possibly renting computers and software.
- Export results using a postprocessor. This allows converting results to many established file formats like: CSV, PHASE, Sweep, PLINK, EIGENSOFT, Structure, Arlequin or DnaSP.
- Linux and Windows versions for 32 and 64 bit platforms.

6.2.1 Human Genome Diversity Project (HGDP) database

The HGDP was originally a DNA bank created with the intention to represent (most of) the entire human population. Around 2005, the DNA was sequenced using genome wide DNA chips from Illumina containing 650,000 markers. The dataset is in the public domain and was used in our experiments. In [160] the project is described as:

"The Human Genome Diversity Project (HGDP) was started by Stanford University's Morrison Institute and a collaboration of scientists around the world. It is the result of many years of work by Luigi Cavalli-Sforza, one of the most cited scientists in the world, which has published extensively in the use of genetics to understand human migration and evolution. The HGDP data sets have often been cited in papers on such topics as population genetics, anthropology, and heritable disease research.

The project has noted the need to record the genetic profiles of endogenous populations, as isolated populations are the best way to understand the genetic frequencies that have clues into our distant past. The relationship between such populations allows inferring the humankind journey from the initial humans that left Africa and populated the world. The HGDP-CEPH Human Genome Diversity Cell Line Panel, is a resource of 1063 cultured lymphoblastoid cell lines (LCLs) from 1050 individuals in 52 world populations, banked at the Foundation Jean Dausset-CEPH in Paris."

The dataset contains a genotype file with the 650,000 markers for each of the 1043 individuals in the study and the phenotype is the population to which the individual belongs. The list of all populations can be found in table 5.12.

6.2.2 PSExplorer, a standalone GEM implementation

PSExplorer [6] is a standalone GEM-based application distributed with the HGDP database and with gene, functionality and GWAS files up to date at the time of its publication. The application also includes n-factor functionality like the algorithms described in chapter 5 for trial. The program is useful for analyzing population stratification which is important in GWAS studies as listed in the NCI workgroup recommendations (see table 5.11). The reason why population stratification is important is explained above in 5.4. The program is distributed as free software.

6.3 The WLS library

Besides implementing WLS [3] in GoKnot, an open source implementation written in C++, Pascal and Java was written by the author to promote the idea to the widest possible audience. The web page [91] includes a description of WLS including, source code to build the lookup tables, a program to test the data built in the table for verification and instructions on implementing WLS.

The program is open source, distributed under a Berkeley-type license [161]. This means, it can be included in both open and close source programs.

6.4 Implementation of MCTS to the Strip Packing Problem

This dissertation also mentions an implementation of MCTS to the Strip Packing Problem done by the author in 2008. The work is described in an unpublished paper in Spanish [33]. This implementation is one of the two implementations in optimization problems described in this dissertation. All the software written for the implementation is released as open source and can be found in the same website as the paper.

6.5 Prototype/scripting software used

Besides all the software mentioned above, some prototype level software was used mainly for data analysis:

- NMF of M-eval described in 3.4 was done in Matlab [162] using third party code [80].
- Elo rating and confidence intervals, as well as Wilcoxon matched pairs tests for assessing the difference in computed times used were performed by Statistica 8.0 [163] Visual Basic scripts.
- All tables with experimental results in chapters 4 and 5 were created automatically using Statistica 8.0 Visual Basic scripts.
- Data analysis described in chapter 3 and 4.5 was done using Statistica 8.0

Chapter 7. Findings and discussion

In this final chapter we summarize our findings and conclude. But first, we want the reader to note the relevance of what has been happening in MCTS during the last five years, the same period of the work described in this dissertation. We describe only the most important applications in best known problems and games to put into perspective the importance of MCTS research. We have been researching in MCTS from its very beginning, when the whole algorithm was still called UCT, and have been exchanging preliminary results and ideas with other researchers in [19] while they were doing the research that has seeded the field. Five years later, the success of MCTS in most fields comes as no surprise to us who have been watching intelligent behavior arise from stochastic experimentation for five years. Most of the first section is taken from an extensive survey [85] which cites almost 250 articles in peer reviewed journals describing MCTS research in the last five years, completed with other relevant research results in some well known games and a very interesting work on Beam MCTS.

7.1 MCTS's success story: Preeminent algorithm in many hard problems

The mentioned MCTS survey [85] is very extensive describing in almost 50 pages the different published MCTS variations in many fields. It supersedes all previously known descriptions of application of MCTS in different fields. It also includes two tables describing which of 66 published MCTS variants/heuristics have been implemented in 69 games/problems. In this section, we exclude implementations in computer *go* which have already been addressed in this dissertation and focus on "best known" games and problems citing a maximum of two works per problem, usually the first one and the (considered) strongest. It is also worth

noting that no MCTS implementations have been described yet in the fields of neither human genetics nor biology, making our work a pioneer in the fields.

7.1.1 MCTS in two player and multiplayer games

Non-randomized two player board games:

- **Hex**: This first MCTS implementation dates from 2008 and the program also plays the games: Y, *Star and Renkula! [164] The current best program is also MCTS and RAVE based, named MOHEX described in [165].
- **Havannah**: A UCT Havannah player with AMAF and a playout horizon was implemented in 2009 [166],
- **Lines of Action:** An MCTS-Solver is able to prove the game-theoretic values of positions given sufficient time was implemented in 2008 [167].
- **Othello** (Reversi): Nijssen developed a UCT player for Othello called MONTHELLO in 2007 [168].
- **Amazons:** Amazons was researched by the UCT creator Levente Kocsis. He proved the superiority of plain UCT over flat Monte Carlo for Amazons in 2006 in his seminal paper [169].
- **Arimaa:** The implementation of a UCT player in Arimaa was researched in Kozelek's master thesis in 2009 [170].
- **Shogi**: Sato et al. describe a UCT Shogi player with a number of enhancements was first described in 2009. The original paper is in Japanese and was translated for the ICGA journal in 2010 [171]. Shogi has been a very active research area in Japan for decades and at the time of this publication top programs were playing approximately at professional human level. Top programs are based on minimax search with α—β pruning and domain specific ordering and pruning heuristics. The MCTS implementation was not top level in Shogi.
- **Mancala** (by other names: Oware, Wari and Awari): Even if this game is already solved since 2003 [172], a UCT player for Mancala and minimax search both perform at similar high level according to [173].
- **Chinese checkers**: Nijssen and Winands applied their Multi-Player Monte-Carlo Tree Search solver (MPMCTS-solver) and progressive history techniques to Chinese checkers, in 2011 [174].
- **General Game Playing**: General Game Playing [34] has been, like *go*, another area in which MCTS has been dominant producing an "MCTS era" in which all strong programs are MCTS

based. The first MCTS program was CADIAPLAYER developed by Hilmar Finnsson for his master's thesis in 2007 [175]. Current MCTS players include ARY [176] and CENTURIO [177].

Randomized games:

- **Poker**: MCTS is currently state of the art in Poker [178]. The Monte-Carlo playout models the opponents' strategies implemented using Expectimax tree search. Computer poker has also experienced a revolution in recent years. In 2006 a group of researchers including Michael Bowling stated in a conference (see slide 16 in [179]) *"Expectimax cannot be used for Poker"*. Nowadays, the team from University of Alberta Computer Poker Research Group, of which Michael Bowling is part, is currently top level. According to "The annual AAAI Computer Poker Competition": Their program, Hyperborean, was gold and bronze in the two categories of Heads-up Limit Texas Hold'em, in 2011, and silver in 2012. Difficulties have been "ironed out" [180] and extremely large trees are stored in the form of LUTs stored in clusters of computers. Furthermore, Monte Carlo Counter Factual Regret (MCCFR) has been able approximate Nash equilibria and the notion of **exploitability** has been coined and is measurable [181]. In [182] it is stated that exploitability has been reducing in computer Poker over the recent years. If, in the near future, exploitability (measured in milliblinds per game) reached near zero values for some program, it would mean that no long term strategy exists consistently winning any money against it. This is arguably the closest possible situation to Poker being "solved" (as randomized hidden information games can obviously not be minimax-solved) and that could happen in the next three years.
- **Backgammon**: The best Backgammon programs are stronger than the best human players. The UCT-based player MCGAMMON developed in 2007 [183] is the only implementation mentioned in the survey and it did not reach top level.

7.1.2 MCTS in optimization problems

MCTS implemented in one player games (puzzles, solitaires):

- **Bubble Breaker** (and other similar puzzles): MCTS was implemented for Bubble Breaker in [184]. Beside the successful implementation, the paper introduces Beam MCTS a very promising

algorithm for deep optimization problems. Beam MCTS is included in our future work on the GEM framework in 7.3.1.

- **Morpion Solitaire**: Morpion is an NP-hard solitaire puzzle in which the player successively colors vertices of an undirected graph. The aim is to make as many moves as possible. Reflexive Monte Carlo Search solved the puzzle in 78 moves, beating the existing human record of 68 moves and previous AI record of 74 moves [185]. Later, the same author used Nested Monte Carlo Search and improved the solution to 80 moves. Unfortunately, we were unable to find the paper reporting this improvement cited in the MCTS survey, but the same author describes Nested Monte Carlo Search in [176].
- **Sudoku** (and also Kakuro): Sudoku, is the NP-complete (versus board size) logic puzzle. MCTS was applied to the 16x16 Sudoku (also Nested MCTS).

MCTS implemented in real-time one player games (videogame):

- **Pac-Man**: There are competitions and a development platform to play (both players: the human player and the machine). The annual CIG conference hosts such events and conference papers are presented regularly, for example [186]. MCTS is currently the best algorithm for both players.

Implementations other than games:

- **Security** (biometrics): MCTS methods were used to evaluate vulnerability to attacks in an image based authentication system. The results obtained are promising and suggest a future development of an MCTS based algorithm [187].
- **Mixed Integer Programming**: UCT is applied to guide Mixed Integer Programming (MIP), comparing the performance of the UCT based node selection with that of CPLEX, a traditional MIP solver, and best-first, breadth-first and depth-first strategies [188].
- **Travelling Salesman Problem**: The TSP with time windows using Nested Monte Carlo Search was studied in [189]. The annual CIG also host real time TSP competitions in which MCTS is also the dominating algorithm.
- **Sailing Domain**: UCT was also applied to the sailing domain by Kocsis and Szepesvari in their seminal paper [20]. The Stochastic Shortest Path (SSP) problem describes a sailboat searching for the shortest path between two points under fluctuating wind conditions.
- **Physics Simulations**: The simulation of many problems in physics using Hierarchical Optimistic Optimization applied to Trees (HOOT) algorithm [190].

- **Function approximation**: The approximation of Lipschitz functions *(I.e., functions satisfying a strong form of uniform continuity: For every pair of points on the graph of this function, the absolute value of the slope of the line connecting them is no greater than a definite real number named the Lipschitz constant.)* is used to compare Bandit Algorithm for Smooth Trees (BAST) (an early UCT variant described in [191]) with flat UCB. Rimmel also describes MCTS applied to function approximation in his PhD dissertation [192].
- **Constraint Problems**: A real-time algorithm based on UCT to solve quantified constraint satisfaction problems (QCSP). Vanilla UCT was not outperforming other algorithms, but the authors developed a constraint propagation technique that allows the tree to focus in the most promising parts of the search space [193].
- **Scheduling Problems**: Results in the 4th International Planning Competition (IPC-4) showed MCTS based Monte Carlo Random Walk (MRW) planner with promising results compared to the other planners tested, including FF, Marvin, YASHP and SG-Plan [194]. Also [195] combined UCT with heuristic search in the Mean-based Heuristic Search for anytime Planning (MHSP) method.
- **Production Management Problems**: PMPs are planning problems that require a parameter optimization process. An MCTS algorithm was used to solve PMPs, getting results faster than Evolutionary Planning Heuristics (EPH), reaching at least the same score in small problems and outperforming EPH in large problems [196].
- **Bus Regulation Problem**: The BRP is scheduling bus waiting times to minimize delays for the passengers. Nested Monte Carlo search with memorization outperformed the other methods tested in [197].

7.2 Findings

7.2.1 Our research question

This PhD thesis did not start with a unique question to be researched, something like "Does the use of expert knowledge improve MCTS?". The answer would have been easy: Of course, it does. Search is always about finding knowledge guiding the search in promising directions. Most authors have already stated that, although it is worth noting how well knowledge-less MCTS makes intelligent behavior emerge naturally in simple problems like the SPP. But we decided to implement what is arguably one of the hardest possible

implementations in game research and possibly in AI research, *go*. Besides the learning curve of the game itself, a look at table 6.18 gives an idea on the amount work done on our system. We did it without taking any shortcuts: implementing everything (including the testing framework) and researching at each step rather than just implementing what had been described as most successful. The implementation culminates our ten year long period in computer *go* (2002-2012), half of it dedicated to MCTS. This dissertation only covers the positive results, with the exception of what is described in 6.1.3. On that issue, we were on the wrong path more than one year (erroneously) sure that good move prediction rates would result in strong evaluation in the simulations.

Just like *go* was once part of Japanese (male) education as a way of learning about strategy, but also about improvement through intense dedication to abstract thinking, what justifies the PhD thesis is the path itself rather than answering some predetermined questions. When this started, expert knowledge was mainly intended to mean "knowledge learned from human master level players". Some of the ideas worked fine, most did not and we grasped the importance of knowledge learned from "human *go* teachers": teaching correct answers like all the heuristics described, especially in chapter 4. Furthermore, MCTS is about online learned knowledge and we finally succeeded in extending that online learning to the playout in the form of a loose tree, which we consider our main contribution in computer *go*, finally completing our *"way of go"*.

As if computer *go* was not hard enough, we implemented MCTS in human genetics. From the beginning we have been stating that common practice in genetic etiology, cumulative risk models, is very far from being the best option in genotype/phenotype prediction models. We have proven experimentally that cumulative risk models performed worst in all problems, for all times and for all model sizes. In this case, we had proven our point if verifying a predetermined hypothesis had been our aim. As in *go*, the implementation itself is far more important than this conclusion, which is recently getting attention coined as "high order SNP combinations" [39]. Our n-factor GEP is, of course, a high order SNP combination. As in *go*, the hard work implementing GEM has provided: further insight in MCTS applied to optimization problems, an unexpected finding and the framework itself which has a potential for many applications. Again, a *"way of go"* in computer genetics.

7.2.2 Contributions to the analysis of MCTS problems

The main contributions of our work in terms of MCTS analysis and problem status are:

- The mathematical analysis including a Wiener process described in 4.1.1. To our knowledge no similar analysis has been made about simulation functions in neither MCTS nor Monte-Carlo literature. This process plays a major role understanding how MCTS scales, for instance when board size in changed. In the beginning of UCT *go*, around 2007, a scalability study had determined the playing strength expected from each doubling in the number playouts. Doubling the board size was expected to result in the program weakening by a factor of 4 (two doublings) due to the increase in width and slowing down by another factor of 4 due to the increase in playout length. The results obtained were even worse before 19x19 specific heuristics were developed (proximity heuristics and progressive widening mainly). The reason of this unexplained weakening was discussed in [19]. The author of this dissertation pointed out that the "missing factor" was of the size predicted by the increase of variance in the Wiener process resulting in bias towards 1/2 as a result of the increase in playout length. Sylvain Gelly, the main author of Mogo agreed that this was the plausible explanation of the weakening. No other model explains the role of the length of the playouts.
- Although controversially in the beginning, the authors have been the first pointing out that the opening is a weakness in 19x19 *go*, why it is a weakness and how this is recognized by players. This subject is described in 3.3.3.
- In our application in human genetics, besides our implementations for the n factor GEP, the problem itself is a major contribution. Before our work, researchers did not have a comparable reference point. All studies have the limitation of proposing a method and implementing it on a different dataset making comparison between methods impossible. Our implementation provides: a dataset (6.2.1), a classification method (both as a mathematical description 5.5.2 and as source code [6]) and easily reproducible results of our algorithms (5.8) using both. This allows using the whole as an open context allowing other authors to compare method-to-method. To our knowledge, no such competition framework exists.

7.2.3 Contributions based on successful implementations of MCTS

In a nutshell, since these contributions have already been described extensively:

- M-eval. (3.4) A solution to improve both style and strength in 19x19 openings.
- WLS. (4.5) An open source tool implemented in learning playouts and also with a potential for being used in a wide range of simulation related areas.

- Learning playouts. (4.7 and 4.8) An important improvement that works on an already strong program and makes an elegant idea work.
- The implementation of MCTS in human genetics. (5.6 and 5.8)

Of course, in all cases the finding is that the fully described and reproducible method works and what improvement that can be expected from implementing it.

7.2.4 Contributions in human genetics

In order to prove the power of the methods proposed in Chapter 5, we applied them to a genome-wide dataset of 640K SNPs genotyped in 944 unrelated individuals from 52 populations from the global sample from the Human Genome Diversity Project-Centre d'Etude du Polymorphisme Humain (HGDP-CEPH) panel [121], to explore the ability of these SNPs across the genome to infer ancestry, and demonstrated that the combination of as few as 12 SNPs bears substantial information about biogeographical ancestry to allowing classifying>85% samples in the correct population without taking into account any prior pre-assignment of individuals to populations. We also provide the first tool for fast, user-friendly automated AIMs selection for efficient ancestry inferences across worldwide populations.

It must be noted that the AIMs sets provided are aimed to maximize information about individual ancestry at minimal cost and sample necessity. It should be stressed that larger sets of markers or a set of highly specialized ones would be required to distinguish the ancestries of genetically similar populations and/or to fully control false positive and false negative results due to population stratification in association studies [198]. However, when genome-wide genotyping data is not available, they would be useful for reducing population stratification effects in case-control association studies by identifying outlier individuals in study samples and perform association tests across ancestry matched individuals.

We envision that these AIMs and the tool will be of particular value in several scenarios:

a. when aiming to infer ancestry and low sample quantity is available (typical in forensics) [199]
b. helping improving the design of disease-association studies at reduced costs by their use for pre-selecting population members before more thorough genome-wide studies are conducted.

7.2.5 Limitations of this study

In our *go* research, since the aim was implementing a strong program, something we finally succeeded at, and we invested five years in the MCTS research using our own code base which started in 2002, we are not aware of any limitations other than not having researched in cluster parallelization. This was both due to material limitations and to the fact that we always keep the intention of producing a commercial program and commercial programs target average users and lightweight platforms like tablets rather than clusters. Not having a cluster version of the program gives us an unfair comparison in competitions, especially since our typical hardware is an off-the-shelf low cost multi-core PC while most of our opponents use clusters and high end multi-core machines.

In our human genetics study, the main limitation is the lack of a comparison with something other than our three algorithms, one of them being the cumulative risk model based on established statistics. It was too easy to just beat this model, so the study lacks comparison with any CPU intensive algorithm other than GrQS. It is worth noting that GrQS is a very well performing algorithm, but with a different classification function, a General Linear regression Model (GLIM) in the problem of finding out which is the best n factor GLIM. It is not established if GrQS is a strong opponent in the n factor GEP or not. As mentioned in 7.2.2, it would be an important step forward if our framework (or some other framework) could be used to make fair comparisons. It is also difficult to have access to genome wide data, since medical ethic procedures make it harder today than it was when HGDP was released. This is a problem even for anonymized data, since complete genotype could be used for identification. Our implementation would have benefited of more subjects and different datasets for the different problems if that had not been a limitation.

7.2.6 Unexpected finding: Possible lack of convergence to optimality

In very wide but not deep applications, like the n factor GEP, some attention must be paid to search paths not containing randomization. Unlike in two players games, where these paths automatically convert MCTS in a minimax algorithm, since the result (always a win for the same player) is extremely good for one player who will favor exploitation and extremely bad for the other who will favor exploration and find refutation paths

should any exist. But in optimization problems when a good solution is found using a non randomized path and the solutions in its neighborhood are not as good, it results in the algorithm exploring the same solution again and again.

Of course, from a theoretical point of view the algorithm would not be completely "trapped" by the solution as, when the number of visits increases, the algorithm still explores alternatives. But for practical reasons it doesn't work. Firstly, the exploration is designed to work when the leaves represent classes of solutions combining the path in the tree with a randomized simulation, not one solution evaluated repeatedly. Secondly, the siblings are not good enough to beat the solution and are also explored repeatedly, since they are also non randomized (this applies in the GEP implementation, since all paths containing the same number of SNPs are non-randomized). As a result, even if theoretically the algorithm would "someday" explore any given path, in practice it is flawed because not only it is evaluating the same solution repeatedly, but its inefficiency also increases *ad infinitum* since the alternatives are also explored repeatedly.

We worked around this problem by biasing the value propagated in the tree when the path is non-randomized. The bias was small (like 2% less than the real value), but still reduced the number of repeated evaluations to low levels without much impact on the search. We did not research the subject in depth considering it tolerable when the search "escaped" the trap after not many simulations focusing on new promising targets. This solution is good enough not to be considered a limitation, but we mentioned another idea as future work below in 7.3.1.

7.3 Discussion

In conclusion, we are excited about having been a part of a success story in computer science from its beginning, MCTS research. MCTS is a XXI century algorithm already playing an important role in many fields, and that role is probably going to increase in the future. One reason for this is: MCTS is a skeleton that supports diverse ideas for including knowledge, both in the tree and in the simulations, and hybridization with other algorithms.

We did not want to present this dissertation as a mere collection of results, but have structured the main body as a complete work that can be used as a post degree textbook. Chapters 3 and 4, possibly without the final *go* specific descriptions if the reader is not interested in computer *go*, provide insight on questions a new MCTS researcher should ponder about after she has read the basic descriptions. Chapter 5 is a fully self

contained description of an actual problem in biology including everything a computer scientist needs to know to understand the field. Besides the already mentioned (7.2.2 and 7.2.3) contributions to the field, our research has produced quality software both in computer *go* and in genetics than will hopefully be used in new research and in useful products.

7.3.1 Future work

A number of ideas could be the next steps both in genetics and for the *go* engine. The lowest hanging fruits could be:

- In genetics: Make MCTS automatically find parameter settings resulting in correct exploration metrics. Only two parameters: the K in UCT and the initial exploration width are the initial targets. Since the response is not in {0, 1} but an integer value (the number of well classified individuals), some sampling could be used to estimate its distribution. An estimate of the number of simulations that will be allowed to run and some testing runs can also find correct exploration width settings. This could make the algorithm more robust over a wide range of applications.
- In genetics: Explore alternatives to the method avoiding MCTS repeating the same path (7.2.6) actually using the penalization described in 5.6. Really pruning leaves that lead to non randomized paths by completely removing them from the tree. This would solve the problem completely and without interfering with the algorithm by propagating inaccurate results.
- In genetics: Implement Beam MCTS [184] to widen the range of models to higher numbers of SNPs like 50 SNP models. These models could be interesting in forensics to assess classification with high certitude. Our current approach is very wide but not deep. Beam MCTS is a promising idea to make the program support simultaneously wide and deep searches.
- In *go*: Implement another layer of WLS that generates the moves just as described in 4.7 but is not updated in the playout. The values in the array of WLS are updated while the engine is pondering (the opponent is thinking) from the knowledge in the tree from the previous search. If this idea worked, it would be a way of propagating knowledge learned in the tree to the playouts, a long time dreamed idea in the computer *go* community no author has yet reported success in.

Chapter 8. Conclusions / Conclusiones

8.1 Conclusions

In this section we recapitulate major conclusions that have already been discussed along this dissertation as a bulleted list. These conclusions do not include other contributions of our research to the field such as: novel ideas, free software or research frameworks. We only include methodologically sound conclusions obtained from the experimental sections.

- M-Eval, as implemented in 2009, achieves a 16 point advantage in the first 50 moves of a 19x19 *go* game against the same program without M-Eval.
- M-Eval, as implemented in 2011, achieves an 80 Elo point improvement over a complete 19x19 *go* game against the same program without M-Eval.
- The Jump to Past State (JPS) heuristic solves the saturation problem in Win/Loss States (WLS).
- The settling time is a linear function of the end of scale in WLS.
- The auto-balancing policy described in 4.7.4 improves the strength of the learning policy.
- The learning playout policy described in 4.7.5 achieves at least a 90 Elo point improvement for all board sizes tested and all CPU allocations tested against the same program playing the same number of simulations with the base policy.

- The learning playout policy outperforms the base policy against a reference opponent by 128 Elo points given the same number of simulations for 13x13 go.
- The non parametric, cross validated classification method described in 5.5.2 is capable of achieving high rates (above 80%) of correct classification in complex problems (classifying individuals in 7 categories) given appropriate SNPs.
- Algorithms for searching high order SNP combinations using search outperform lists of best classifiers in all problems and all numbers of SNPs.
- MCTS outperforms the other methods in each of the problems with more than 4 SNPs.
- MCTS benefits from the extra CPU allocation, making it suitable for deep searches in large genetic searches.

8.2 Conclusiones

En esta sección recapitulamos las principales conclusiones que ya han sido discutidas a lo largo de esta tesis simplemente enumerándolas. Estas conclusiones no incluyen otras contribuciones de nuestra investigación al campo de la misma como: ideas novedosas, software libre o entornos de investigación. Solamente incluimos conclusiones metodológicamente válidas obtenidas de los experimentos.

- M-Eval, tal y como fue implementado en 2009, alcanza una ventaja de 16 puntos en las 50 primeras jugadas de una partida *go* 19x19 contra la misma versión del programa sin M-Eval.
- M-Eval, tal y como fue implementado en 2011, alcanza una ventaja de 80 puntos Elo en partidas completas de go 19x19 contra la misma versión del programa sin M-Eval.
- La heurística JPS (*Jump to Past State*) resuelve el problema de la saturación de los WLS (*Win/Loss States*).
- El *tiempo de establecimiento* es una función lineal del parámetro *fin de escala* de un WLS.
- La política de equilibrado del número de jugadas descrita en 4.7.4 contribuye a la mejora de la política de aprendizaje *online* en simulaciones.
- La política de aprendizaje *online* en simulaciones descrita en 4.7.5 alcanza al menos 90 puntos Elo de mejora para todos los tamaños de tablero probados y todas las asignaciones de CPU probadas contra el mismo programa realizando el mismo número de simulaciones con la política de base.

- La política de aprendizaje *online* en simulaciones supera a la política de base contra un adversario de referencia por 128 puntos Elo dado en mismo número de simulaciones en *go* 13x13.
- El método de clasificación no paramétrico con validación cruzada descrito en 5.5.2 puede alcanzar altas tasas de clasificación correcta (más del 80%) en problemas complejos (clasificar en 7 categorías) dándole un conjunto de SNPs apropiado.
- Los algoritmos para encontrar conjuntos de SNPs basados en búsqueda superan a los basados en listas de mejores clasificadores en todos los problemas con todos los tamaños.
- MCTS supera a todos los demás métodos en todo los problemas con más de 4 SNPs
- MCTS se beneficia de una mayor asignación de CPU, lo que lo convierte en apropiado para búsquedas profundas en aplicaciones genéticas.

References

1. Basaldúa, J., T.N. Yang, and J.M. Moreno-Vega. *M-eval: A multivariate evaluation function for opening positions in computer Go*. in *International Workshop on Computer Games (IWCG 2010), Hsinchu (Taiwan)*. 2010. IEEE conference.
2. Basaldúa, J. and J.M. Moreno-Vega. *Advances in M-eval: A Multivariate Evaluation Function for Opening Positions in Computer Go*. in *2012 International Go Symposium*. 2012. U.S. Go Congress in Black Mountain, North Carolina.
3. Basaldúa, J. and J.M. Moreno-Vega. *Win/Loss States: An efficient model of success rates for simulation-based functions*. in *CIG 2012*. 2012. Granada, Spain, pp. 41-46.
4. Basaldúa, J., S. Stewart, J.M. Moreno-Vega, and P. Drake, *Two Online Learning Playout Policies in Monte-Carlo Go: An application of Win/Loss States.* IEEE Transactions on Computational Intelligence and AI in Games (under review), 2013.
5. Pino-Yanes, M., A. Corrales, S. Basaldua, A. Hernandez, L. Guerra, J. Villar, and C. Flores, *North African influences and potential bias in case-control association studies in the Spanish population.* PLoS One, 2011. 6(3): p. e18389.
6. Basaldúa, J., J.M. Moreno-Vega, and C. Flores, *A combinatorial approach for selecting a minimum set of ancestry informative markers for continental assignation of human DNA samples.* (working paper), 2013.
7. American, G.A. *What Is Go?* 2012; Available from: http://www.usgo.org/what-go
8. British, G.A. *About the Game*. 2012; Available from: http://www.britgo.org/about.html
9. Mori, H. *The Interactive Way To Go*. 2012; Available from: http://playgo.to/iwtg/en/
10. Cho, C., *GO: A complete introduction to the game*1997: Kiseido Publishing Co.
11. Wikimedia Foundation, I. *Go (game)*. 2012; Available from: http://en.wikipedia.org/wiki/Go_(game)
12. (collaborative). *Sensei's Library*. 2012; Available from: http://senseis.xmp.net/
13. Chinese, W.A. *Rules of Weiqi (Go)*. 2002; Available from: http://home.snafu.de/jasiek/c2002.pdf
14. Nihon, K. *The Japanese Rules of Go*. 1989; Available from: http://www.cs.cmu.edu/~wjh/go/rules/Japanese.html
15. van der Werf, E., *AI techniques for the game of Go*, 2005, Maastricht Univ., : Maastricht, The Netherlands.
16. Müller, M., *Computer go.* Artificial Intelligence, 2002. 134(1): p. 145-179.
17. Wikimedia Foundation, I. *Computer Go*. 2012; Available from: http://en.wikipedia.org/wiki/Computer_Go
18. Alberta-University. *Computer Go Bibliography*. 2012; Available from: http://webdocs.cs.ualberta.ca/~games/go/compgo_biblio/compgo_biblio.html
19. (collaborative). *Computer Go Mailing List*. 2012; Available from: http://computer-go.org/mailman/listinfo/computer-go
20. Kocsis, L. and C. Szepesvári, *Bandit based monte-carlo planning.* Machine Learning: ECML 2006, 2006: p. 282-293.

21. Gelly, S. and D. Silver. *Combining online and offline knowledge in UCT*. in *Proceedings of the International Conference of Machine Learning (ICML 2007)*. 2007. ACM p. 273-280.
22. Gelly, S., Y. Wang, R. Munos, and O. Teytaud, *Modification of UCT with patterns in Monte-Carlo Go.* Technical report, INRIA RR-6062, 2006.
23. Gelly, S. and D. Silver, *Monte-Carlo tree search and rapid action value estimation in computer Go.* Intelligence (AIJ) 175:1856–1875, 2011.
24. Coulom, R. *Computing Elo ratings of move patterns in the game of Go*. in *Proceedings of the Computers Games Workshop 2007 (CGW 2007), p. 113-124*. 2007.
25. Chaslot, G., M.H.M. Winands, H. Herik, J. Uiterwijk, and B. Bouzy, *Progressive strategies for Monte-Carlo tree search.* New Mathematics and Natural Computation, 2008. 4(3): p. 343.
26. (collaborative). *UCT for Monte Carlo computer go* 2007; Available from: http://senseis.xmp.net/?UCT
27. Browne, C.R., Philipp; Powley, Edward *Everything Monte Carlo Tree Search* 2012; Available from: http://www.mcts.ai/
28. Elo, A.E., *The rating of chessplayers, past and present*1986: 2nd Edition. New York: Arco.
29. Eppstein, D. *Computational Complexity of Games and Puzzles*. 2012; Available from: http://www.ics.uci.edu/~eppstein/cgt/hard.html
30. Zobrist, A.L., *A new hashing method with application for game playing.* ICCA Journal, 1970. 13(2): p. 69-73.
31. Benson, D.B., *Life in the game of Go.* Information Sciences, 1976. 10(2): p. 17-29.
32. (collaborative). *Ing Prize*. 2012; Available from: http://senseis.xmp.net/?IngPrize
33. Basaldúa, J. *UCT applied to the 2D Strip Packing Problem (SPP)*. 2008; Available from: http://www.dybot.com/research/doku.php?id=uct_for_the_spp
34. Wikimedia Foundation, I. *General game playing*. 2012; Available from: http://en.wikipedia.org/wiki/General_Game_Playing
35. Stanford-University. *General game playing*. 2012; Available from: http://games.stanford.edu/
36. Stanford-University. *Game Description Language Specification*. 2008; Available from: http://games.stanford.edu/language/spec/gdl_spec_2008_03.pdf
37. Macready, W.G. and D.H. Wolpert, *Bandit problems and the exploration/exploitation tradeoff.* Evolutionary Computation, IEEE Transactions on, 1998. 2(1): p. 2-22.
38. Dyckhoff, H., *A typology of cutting and packing problems.* European Journal of Operational Research, 1990. 44(2): p. 145-159.
39. Fang, G., M. Haznadar, W. Wang, H. Yu, M. Steinbach, T.R. Church, W.S. Oetting, B. Van Ness, and V. Kumar, *High-order SNP combinations associated with complex diseases: efficient discovery, statistical power and functional interactions.* PLoS One, 2012. 7(4): p. e33531.
40. Cavalli-Sforza, L.L., *The human genome diversity project: past, present and future.* Nature Reviews Genetics, 2005. 6(4): p. 333-340.
41. Rosenberg, N.A., *Standardized subsets of the HGDP-CEPH Human Genome Diversity Cell Line Panel, accounting for atypical and duplicated samples and pairs of close relatives.* Ann Hum Genet, 2006. 70(Pt 6): p. 841-7.
42. Basaldúa, J. *Complementary materials to M-eval*. 2012; Available from: http://www.dybot.com/research/doku.php?id=m-eval_materials
43. Wang, Y. and S. Gelly. *Modifications of UCT and sequence-like simulations for Monte-Carlo Go*. in *Procedings of the 2007 IE Symposium on Computational Inteligence and Games (CIG07)*. 2007. Hawai, USA, p. 175-182.

44. Silver, D., R. Sutton, and M. Müller. *Reinforcement learning of local shape in the game of Go*. in *Proceedings of the 20th IJCAI*. 2007. p. 1053–1058.
45. Coulom, R., *Criticality: a monte-carlo heuristic for go programs.* Invited talk at the University of Electro-Communications, Tokyo, Japan, 2009.
46. Bouzy, B. and B. Helmstetter, *Monte-carlo go developments.* Advances in Computer Games, 2003. 10: p. 159-174.
47. Brügmann, B., *Monte carlo go.* White paper, 1993.
48. Helmbold, D.P. and A. Parker-Wood. *All-Moves-As-First Heuristics in Monte-Carlo Go*. in *Proceedings of the International Conference on Artificial Intelligence*. 2009. Las Vegas, Nevada, 2009, pp. 605–610.
49. Enzenberger, M., M. Muller, B. Arneson, and R. Segal, *FUEGO An Open-Source Framework for Board Games and Go Engine Based on Monte-Carlo Tree Search.* Computational Intelligence and AI in Games, IEEE Transactions on, 2009(99): p. 1-1.
50. Baudis, P. *Pachi: Software for the Board Game of Go / Weiqi / Baduk*. 2012; Available from: http://pachi.or.cz/
51. Kato, H. and I. Takeuchi. *Parallel monte-carlo tree search with simulation servers*. in *In 13th Game Programming Workshop (GPW-08)*. 2008.
52. Cazenave, T. and N. Jouandeau, *A parallel Monte-Carlo tree search algorithm.* Computers and Games, 2008: p. 72-80.
53. Soejima, Y., A. Kishimoto, and O. Watanabe, *Evaluating root parallelization in Go.* IEEE Transactions on Computational Intelligence and AI in Games, 2010. 2(4): p. 278-287.
54. van Niekerk, F., S. Kroon, G.J. van Rooyen, and C.P. Inggs. *Monte-Carlo Tree Search Parallelisation for Computer Go*. in *Proc. of the South African Institute for Computer Scientists and Information Technologists Conference*. 2012. p. 129-138.
55. Chaslot, G., M. Winands, and H. van Den Herik, *Parallel monte-carlo tree search.* Computers and Games, 2008. 5131: p. 60-71.
56. Enzenberger, M. and M. Müller, *A lock-free multithreaded Monte-Carlo tree search algorithm.* Advances in Computer Games, 2010: p. 14-20.
57. Silver, D., *Mathematical details of RAVE (white paper)*, 2008.
58. Ojima, Y. *Zen (go program)*. 2009; Available from: http://senseis.xmp.net/?ZenGoProgram
59. Yamashita, H. *Hiroshi's Computer Shogi and Go*. 2012; Available from: http://www32.ocn.ne.jp/~yss/
60. Fotland, D. *Knowledge Representation in The Many Faces of Go*. in *Second Cannes/Sophia-Antipolis Go Research Day*. 1993.
61. Coulom, R., *CLOP: Confident Local Optimization for Noisy Black-Box Parameter Tuning.* Advances in Computer Games, 2012. volume 7168 of Lecture Notes in Computer Science: p. 146-157.
62. KGS. *KGS Go Server*. 2003; Available from: http://www.gokgs.com/
63. Berlekamp, E., *Introductory overview of mathematical Go endgames.* Combinatorial Games, 1991. 43: p. 73-100.
64. Berlekamp, E., *Thermographs find the biggest move asymptotically.* E. Unpublished manuscript, 1992.
65. Mueller, M., *Computer Go as a sum of local games: an application of combinatorial game theory*, 1995, Swiss Federal Institute of Technology Zürich.
66. Ralaivola, L., L. Wu, and P. Baldi, *SVM and pattern-enriched common fate graphs for the game of Go.* ESANN, 2005: p. 27-29.

67. Hersey, A., N. Sylvester, and P. Drake. *An improved opening book for computer Go*. 2010; Available from: http://college.lclark.edu/live/files/6755
68. Mullins, J., D. Tillis, and P. Drake. *Implementing an opening book in computer Go*. 2009; Available from: thescipub.com/pdf/10.3844/jcssp.2012.1594.1600
69. Hideo, O., *Opening theory made easy: twenty strategic principles to improve your opening game,* 2002: Kiseido Publishing Company.
70. Ishigure, I., *In the beginning : the opening in the game of go*. Elementary go series, 1973, Tokyo: Ishi Press. 152 p.
71. Sakata, E. and R. Bozulich, *Modern Joseki and Fuseki*. 3rd ed. The Ishi Press go series, 2007, Bronx, NY: Ishi Press International.
72. Takagawa, S., *The vital points of Go,* 1969: The Japanese Go Association.
73. Yen, S.J. and S.C. Hsu, *A positional judgment system for computer Go.* Advances in Computer Games, 2001. 9: p. 313-326.
74. Wolf, T., *A Dynamical Systems Approach for Static Evaluation in Go.* IEEE Transactions on Computational Intelligence and AI in Games, 2011(99): p. 1-1.
75. Paatero, P. and U. Tapper, *Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values.* Environmetrics, 1994. 5(2): p. 111-126.
76. Paatero, P., *Least squares formulation of robust non-negative factor analysis.* Chemometrics and Intelligent Laboratory Systems, 1997. 37(1): p. 23-35.
77. Lee, D.D. and H.S. Seung, *Learning the parts of objects by non-negative matrix factorization.* Nature, 1999. 401(6755): p. 788-791.
78. Donoho, D. and V. Stodden, *When does non-negative matrix factorization give a correct decomposition into parts.* Advances in neural information processing systems, 2003. 16.
79. Kullback, S. and R.A. Leibler, *On information and sufficiency.* The Annals of Mathematical Statistics, 1951. 22(1): p. 79-86.
80. Li, Y.N., Alioune. *The Non-Negative Matrix Factorization Toolbox in MATLAB (The NMF MATLAB Toolbox)*. 2011; Available from: http://cs.uwindsor.ca/~li11112c/nmf.html
81. Seung, D. and L. Lee, *Algorithms for non-negative matrix factorization.* Advances in neural information processing systems, 2001. 13: p. 556-562.
82. Knuth, D.E. and R.W. Moore, *An analysis of alpha-beta pruning.* Artificial intelligence, 1976. 6(4): p. 293-326.
83. Hart, P.E., N.J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths.* IEEE Transactions on Systems Science and Cybernetics, 1968. 4(2): p. 100-107.
84. Nagai, A., *Df-pn algorithm for searching AND/OR trees and its applications*, 2002, PhD thesis, Department of Information Science, University of Tokyo.
85. Browne, C.B., E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, *A survey of monte carlo tree search methods.* IEEE Transactions on Computational Intelligence and AI in Games, 2012. 4(1): p. 1-43.
86. Huang, S.-C., R. Coulom, and S.-S. Lin, *Monte-Carlo Simulation Balancing in Practice.* Lecture Notes in Computer Science, 2011. 6515: p. 81.
87. Lin, G. *Fuego Go: The Missing Manual,*. 2009; Available from: http://users.soe.ucsc.edu/~glin/docs/Fuego_Fall09Report.pdf
88. Niu, X., A. Kishimoto, and M. Müller, *Recognizing seki in computer Go.* Advances in Computer Games, 2006. vol 4630 of Lecture Notes in Computer Science, pp. 37–49.

89. Graepel, T., M. Goutrie, M. Krüger, and R. Herbrich, *Learning on graphs in the game of Go.* Artificial Neural Networks—ICANN 2001, 2001: p. 347-352.
90. Nijhuis, E.H.J., *Learning Patterns in the Game of Go*, 2006, M. Sc. Dissertation, Universiteit van Amsterdam.
91. Basaldúa, J. *WLS: open source implementation and complementary materials*. 2012; Available from: http://www.dybot.com/research/doku.php?id=wls_library
92. Agresti, A., *An introduction to categorical data analysis*. Wiley series in probability and mathematical statistics. Vol. 423. 2007: Wiley-Blackwell.
93. Brown, L.D., T.T. Cai, and A. DasGupta, *Interval estimation for a binomial proportion.* Statistical Science, 2001. 16: p. 101-117.
94. Spearman, C., *The proof and measurement of association between two things.* The American journal of psychology, 1904. 15(1): p. 72-101.
95. Silver, D. and G. Tesauro. *Monte-Carlo simulation balancing*. in *ICML '09, Proceedings of the 26th Annual International Conference on Machine Learning,*. 2009. New York, NY, USA, 2009. ACM.
96. Wedd, N. *Human-Computer Go Challenges*. 2011; Available from: http://www.computer-go.info/h-c/index.html
97. Dailey, D. *9x9 Scalability study*. 2008; Available from: http://cgos.boardspace.net/study/index.html
98. Pierce, P.B., *Genetica/Genetics: Un Enfoque Conceptual/a Conceptual Approach,* 2010: Ed. Médica Panamericana.
99. Lander, E.S. and M.S. Waterman, *Calculating the secrets of life: applications of the mathematical sciences in molecular biology,* 1995: Natl Academy Pr.
100. Marchini, J. and B. Howie, *Genotype imputation for genome-wide association studies.* Nature Reviews Genetics, 2010. 11(7): p. 499-511.
101. MedlinePlus. *MedLine Plus : Chronic obstructive pulmonary disease*. NIH National Institutes of Health 2012; Available from: http://www.nlm.nih.gov/medlineplus/ency/article/000091.htm
102. Jakobsdottir, J., M.B. Gorin, Y.P. Conley, R.E. Ferrell, and D.E. Weeks, *Interpretation of genetic association studies: markers with replicated highly significant odds ratios may be poor classifiers.* PLoS Genet, 2009. 5(2): p. e1000337.
103. Pepe, M.S., H. Janes, G. Longton, W. Leisenring, and P. Newcomb, *Limitations of the odds ratio in gauging the performance of a diagnostic, prognostic, or screening marker.* Am J Epidemiol, 2004. 159(9): p. 882-90.
104. Kraft, P. and D.J. Hunter, *Genetic risk prediction--are we there yet?* N Engl J Med, 2009. 360(17): p. 1701-3.
105. Hardy, J. and A. Singleton, *Genomewide association studies and human disease.* N Engl J Med, 2009. 360(17): p. 1759-68.
106. Studies, N.-N.W.G.o.R.i.A., S.J. Chanock, T. Manolio, M. Boehnke, E. Boerwinkle, D.J. Hunter, G. Thomas, J.N. Hirschhorn, G. Abecasis, D. Altshuler, J.E. Bailey-Wilson, L.D. Brooks, L.R. Cardon, M. Daly, P. Donnelly, J.F. Fraumeni, Jr., N.B. Freimer, D.S. Gerhard, C. Gunter, A.E. Guttmacher, M.S. Guyer, E.L. Harris, J. Hoh, R. Hoover, C.A. Kong, K.R. Merikangas, C.C. Morton, L.J. Palmer, E.G. Phimister, J.P. Rice, J. Roberts, C. Rotimi, M.A. Tucker, K.J. Vogan, S. Wacholder, E.M. Wijsman, D.M. Winn, and F.S. Collins, *Replicating genotype-phenotype associations.* Nature, 2007. 447(7145): p. 655-60.

107. Weir, B.S., *Genetic data analysis II : methods for discrete population genetic data*1996, Sunderland, Mass.: Sinauer Associates. xii, 445 p.
108. Rosenberg, N.A., L.M. Li, R. Ward, and J.K. Pritchard, *Informativeness of genetic markers for inference of ancestry.* Am J Hum Genet, 2003. 73(6): p. 1402-22.
109. Barron, A., J. Rissanen, and B. Yu, *The minimum description length principle in coding and modeling.* IEEE Transactions on Information Theory, 1998. 44(6): p. 2743-2760.
110. Kumar, R., M.A. Seibold, M.C. Aldrich, L.K. Williams, A.P. Reiner, L. Colangelo, J. Galanter, C. Gignoux, D. Hu, S. Sen, S. Choudhry, E.L. Peterson, J. Rodriguez-Santana, W. Rodriguez-Cintron, M.A. Nalls, T.S. Leak, E. O'Meara, B. Meibohm, S.B. Kritchevsky, R. Li, T.B. Harris, D.A. Nickerson, M. Fornage, P. Enright, E. Ziv, L.J. Smith, K. Liu, and E.G. Burchard, *Genetic ancestry in lung-function predictions.* N Engl J Med, 2010. 363(4): p. 321-30.
111. Weder, A.B., L. Gleiberman, and A. Sachdeva, *Whites excrete a water load more rapidly than blacks.* Hypertension, 2009. 53(4): p. 715-8.
112. Flores, C., S.F. Ma, M. Pino-Yanes, M.S. Wade, L. Perez-Mendez, R.A. Kittles, D. Wang, S. Papaiahgari, J.G. Ford, R. Kumar, and J.G. Garcia, *African ancestry is associated with asthma risk in African Americans.* PLoS One, 2012. 7(1): p. e26807.
113. Corvol, H., A. De Giacomo, C. Eng, M. Seibold, E. Ziv, R. Chapela, J.R. Rodriguez-Santana, W. Rodriguez-Cintron, S. Thyne, H.G. Watson, K. Meade, M. LeNoir, P.C. Avila, S. Choudhry, and E.G. Burchard, *Genetic ancestry modifies pharmacogenetic gene-gene interaction for asthma.* Pharmacogenet Genomics, 2009. 19(7): p. 489-96.
114. Yang, J.J., C. Cheng, M. Devidas, X. Cao, Y. Fan, D. Campana, W. Yang, G. Neale, N.J. Cox, P. Scheet, M.J. Borowitz, N.J. Winick, P.L. Martin, C.L. Willman, W.P. Bowman, B.M. Camitta, A. Carroll, G.H. Reaman, W.L. Carroll, M. Loh, S.P. Hunger, C.H. Pui, W.E. Evans, and M.V. Relling, *Ancestry and pharmacogenomics of relapse in acute lymphoblastic leukemia.* Nat Genet, 2011. 43(3): p. 237-41.
115. Kersbergen, P., K. van Duijn, A.D. Kloosterman, J.T. den Dunnen, M. Kayser, and P. de Knijff, *Developing a set of ancestry-sensitive DNA markers reflecting continental origins of humans.* BMC Genet, 2009. 10: p. 69.
116. Burchard, E.G., E. Ziv, N. Coyle, S.L. Gomez, H. Tang, A.J. Karter, J.L. Mountain, E.J. Perez-Stable, D. Sheppard, and N. Risch, *The importance of race and ethnic background in biomedical research and clinical practice.* N Engl J Med, 2003. 348(12): p. 1170-5.
117. Burnett, M.S., K.J. Strain, T.G. Lesnick, M. de Andrade, W.A. Rocca, and D.M. Maraganore, *Reliability of self-reported ancestry among siblings: implications for genetic association studies.* Am J Epidemiol, 2006. 163(5): p. 486-92.
118. Rotimi, C.N. and L.B. Jorde, *Ancestry and disease in the age of genomic medicine.* N Engl J Med, 2010. 363(16): p. 1551-8.
119. Campbell, C.D., E.L. Ogburn, K.L. Lunetta, H.N. Lyon, M.L. Freedman, L.C. Groop, D. Altshuler, K.G. Ardlie, and J.N. Hirschhorn, *Demonstrating stratification in a European American population.* Nat Genet, 2005. 37(8): p. 868-72.
120. Berger, M., H.H. Stassen, K. Kohler, V. Krane, D. Monks, C. Wanner, K. Hoffmann, M.M. Hoffmann, M. Zimmer, H. Bickeboller, and T.H. Lindner, *Hidden population substructures in an apparently homogeneous population bias association studies.* Eur J Hum Genet, 2006. 14(2): p. 236-44.
121. Li, J.Z., D.M. Absher, H. Tang, A.M. Southwick, A.M. Casto, S. Ramachandran, H.M. Cann, G.S. Barsh, M. Feldman, L.L. Cavalli-Sforza, and R.M. Myers, *Worldwide human*

relationships inferred from genome-wide patterns of variation. Science, 2008. 319(5866): p. 1100-4.

122. Mountain, J.L. and N. Risch, *Assessing genetic contributions to phenotypic differences among 'racial' and 'ethnic' groups.* Nat Genet, 2004. 36(11 Suppl): p. S48-53.
123. Novembre, J., T. Johnson, K. Bryc, Z. Kutalik, A.R. Boyko, A. Auton, A. Indap, K.S. King, S. Bergmann, M.R. Nelson, M. Stephens, and C.D. Bustamante, *Genes mirror geography within Europe.* Nature, 2008. 456(7218): p. 98-101.
124. Zakharia, F., A. Basu, D. Absher, T.L. Assimes, A.S. Go, M.A. Hlatky, C. Iribarren, J.W. Knowles, J. Li, B. Narasimhan, S. Sidney, A. Southwick, R.M. Myers, T. Quertermous, N. Risch, and H. Tang, *Characterizing the admixed African ancestry of African Americans.* Genome Biol, 2009. 10(12): p. R141.
125. Flores, C., J.M. Larruga, A.M. Gonzalez, M. Hernandez, F.M. Pinto, and V.M. Cabrera, *The origin of the Canary Island aborigines and their contribution to the modern population: a molecular genetics perspective.* Current anthropology, 2001. 42(5): p. 749-755.
126. Tian, C., R. Kosoy, R. Nassir, A. Lee, P. Villoslada, L. Klareskog, L. Hammarstrom, H.J. Garchon, A.E. Pulver, M. Ransom, P.K. Gregersen, and M.F. Seldin, *European population genetic substructure: further definition of ancestry informative markers for distinguishing among diverse European ethnic groups.* Mol Med, 2009. 15(11-12): p. 371-83.
127. Lao, O., K. van Duijn, P. Kersbergen, P. de Knijff, and M. Kayser, *Proportioning whole-genome single-nucleotide-polymorphism diversity for the identification of geographic population structure and genetic ancestry.* Am J Hum Genet, 2006. 78(4): p. 680-90.
128. Bauchet, M., B. McEvoy, L.N. Pearson, E.E. Quillen, T. Sarkisian, K. Hovhannesyan, R. Deka, D.G. Bradley, and M.D. Shriver, *Measuring European population stratification with microarray genotype data.* Am J Hum Genet, 2007. 80(5): p. 948-56.
129. Price, A.L., J. Butler, N. Patterson, C. Capelli, V.L. Pascali, F. Scarnicci, A. Ruiz-Linares, L. Groop, A.A. Saetta, P. Korkolopoulou, U. Seligsohn, A. Waliszewska, C. Schirmer, K. Ardlie, A. Ramos, J. Nemesh, L. Arbeitman, D.B. Goldstein, D. Reich, and J.N. Hirschhorn, *Discerning the ancestry of European Americans in genetic association studies.* PLoS Genet, 2008. 4(1): p. e236.
130. McKeigue, P.M., *Mapping genes that underlie ethnic differences in disease risk: methods for detecting linkage in admixed populations, by conditioning on parental admixture.* Am J Hum Genet, 1998. 63(1): p. 241-51.
131. Seldin, M.F., R. Shigeta, P. Villoslada, C. Selmi, J. Tuomilehto, G. Silva, J.W. Belmont, L. Klareskog, and P.K. Gregersen, *European population substructure: clustering of northern and southern populations.* PLoS Genet, 2006. 2(9): p. e143.
132. Tian, C., R.M. Plenge, M. Ransom, A. Lee, P. Villoslada, C. Selmi, L. Klareskog, A.E. Pulver, L. Qi, P.K. Gregersen, and M.F. Seldin, *Analysis and application of European genetic substructure using 300 K SNP information.* PLoS Genet, 2008. 4(1): p. e4.
133. Kosoy, R., R. Nassir, C. Tian, P.A. White, L.M. Butler, G. Silva, R. Kittles, M.E. Alarcon-Riquelme, P.K. Gregersen, and J.W. Belmont, *Ancestry informative marker sets for determining continental origin and admixture proportions in common populations in America.* Human mutation, 2009. 30(1): p. 69-78.
134. Galanter, J.M., J.C. Fernandez-Lopez, C.R. Gignoux, J. Barnholtz-Sloan, C. Fernandez-Rozadilla, M. Via, A. Hidalgo-Miranda, A.V. Contreras, L.U. Figueroa, P. Raska, G. Jimenez-Sanchez, I.S. Zolezzi, M. Torres, C.R. Ponte, Y. Ruiz, A. Salas, E. Nguyen, C. Eng, L. Borjas, W. Zabala, G. Barreto, F.R. Gonzalez, A. Ibarra, P. Taboada, L. Porras, F. Moreno, A.

Bigham, G. Gutierrez, T. Brutsaert, F. Leon-Velarde, L.G. Moore, E. Vargas, M. Cruz, J. Escobedo, J. Rodriguez-Santana, W. Rodriguez-Cintron, R. Chapela, J.G. Ford, C. Bustamante, D. Seminara, M. Shriver, E. Ziv, E.G. Burchard, R. Haile, E. Parra, and A. Carracedo, *Development of a panel of genome-wide ancestry informative markers to study admixture throughout the Americas.* PLoS Genet, 2012. 8(3): p. e1002554.

135. Paschou, P., J. Lewis, A. Javed, and P. Drineas, *Ancestry informative markers for fine-scale individual assignment to worldwide populations.* J Med Genet, 2010. 47(12): p. 835-47.
136. Lee, E., H.Y. Chuang, J.W. Kim, T. Ideker, and D. Lee, *Inferring pathway activity toward precise disease classification.* PLoS Comput Biol, 2008. 4(11): p. e1000217.
137. Huang, B.E., C.I. Amos, and D.Y. Lin, *Detecting haplotype effects in genomewide association studies.* Genet Epidemiol, 2007. 31(8): p. 803-12.
138. Kim, S. and E.P. Xing, *Statistical estimation of correlated genome associations to a quantitative trait network.* PLoS Genet, 2009. 5(8): p. e1000587.
139. Lahiri, P., *Model selection*. Lecture notes-monograph series2001, Beachwood, Ohio: Institute of Mathematical Statistics. 256 p.
140. Agresti, A., *Categorical data analysis*. 3rd ed. Wiley series in probability and statistics, 2012, Hoboken, NJ: Wiley.
141. Pardoe, I., *Applied regression modeling,* 2012, Hoboken, NJ: Wiley.
142. Hjorth, J.S.U., *Computer intensive statistical methods : validation model selection and bootstrap,* 1999, Boca Raton, Fla.: Chapman & Hall. x, 263 p.
143. Ando, T., *Bayesian model selection and statistical modeling*. Statistics : textbooks and monographs, 2010, Boca Raton: CRC Press. xiv, 286 p.
144. Hayakawa, T., *The likelihood ratio criterion for a composite hypothesis under a local alternative.* Biometrika, 1975. 62(2): p. 451-460.
145. Neyman, J. and E.S. Pearson, *On the problem of the most efficient tests of statistical hypotheses.* Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, 1933. 231: p. 289-337.
146. Bradley, R.A. and M.E. Terry, *Rank analysis of incomplete block designs: I. The method of paired comparisons.* Biometrika, 1952. 39(3/4): p. 324-345.
147. Andersson, A., *Balanced search trees made simple.* Algorithms and Data Structures, 1993: p. 60-71.
148. Basaldúa, J. *GoKnot*. 2002-2012; Available from: http://www.dybot.com/research/doku.php?id=goknot
149. Basaldúa, J. *The Hologram Board System (HBS) assembly language functionality*. 2012; Available from: http://www.dybot.com/research/doku.php?id=hbs_library
150. Basaldúa, J. *The collection of master level games in 19x19*. 2010; Available from: http://www.dybot.com/research/doku.php?id=master_games
151. Basaldúa, J. *GKE: Interfacing go engines with GoKnot*. 2005; Available from: http://www.dybot.com/research/doku.php?id=gke
152. Basaldúa, J. *The Gene Etiology Miner*. 2012; Available from: http://www.biomodelling.eu
153. Hindorff, L.A., H.A. Junkins, J. Mehta, and T. Manolio, *A catalog of published genome-wide association studies.* National Human Genome Research Institute, 2010.
154. Kumar, P., S. Henikoff, and P.C. Ng, *Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm.* Nat Protoc, 2009. 4(7): p. 1073-81.
155. Institute, J.C.V. *SIFT home*. 2010; Available from: http://sift.jcvi.org/

156. Ramensky, V., P. Bork, and S. Sunyaev, *Human non-synonymous SNPs: server and survey.* Nucleic Acids Res, 2002. 30(17): p. 3894-900.
157. Sunyaev, S., V. Ramensky, and P. Bork, *Towards a structural basis of human non-synonymous single nucleotide polymorphisms.* Trends Genet, 2000. 16(5): p. 198-200.
158. Sunyaev, S., V. Ramensky, I. Koch, W. Lathe, 3rd, A.S. Kondrashov, and P. Bork, *Prediction of deleterious human alleles.* Hum Mol Genet, 2001. 10(6): p. 591-7.
159. University, H. *PolyPhen SNP data collection*. 2012; Available from: http://genetics.bwh.harvard.edu/pph/data/
160. Wikimedia Foundation, I. *Human Genome Diversity Project*. 2012; Available from: http://en.wikipedia.org/wiki/Human_Genome_Diversity_Project
161. Wikimedia Foundation, I. *Berkeley Software Distribution*. 2012; Available from: http://en.wikipedia.org/wiki/Berkeley_Software_Distribution
162. MathWorks. *MATLAB, (official webpage)*. 2012; Available from: http://www.mathworks.com/products/matlab/index.html
163. StatSoft. *STATISTICA, Official Product Catalog* 2012; Available from: http://www.statsoft.com/products/statistica-product-catalog/
164. Raiko, T. and J. Peltonen. *Application of UCT Search to the Connection Games of Hex, Y,* Star, and Renkula!* in *Proceedings of the 13th Finnish Artificial Intelligence Conference, STeP 2008, p.89-93*. 2008. Helsinki, Finland.
165. Arneson, B., R.B. Hayward, and P. Henderson, *Monte Carlo tree search in hex.* Computational Intelligence and AI in Games, IEEE Transactions on, 2010. 2(4): p. 251-258.
166. Könnecke, S. and J. Waldmann, *Efficient Playouts for the Havannah Abstract Board Game.* Hochschule Technik, Leipzig, Tech. Rep, 2009.
167. Winands, M., Y. Björnsson, and J.T. Saito, *Monte-carlo tree search solver.* Computers and Games, 2008: p. 25-36.
168. Nijssen, J., *Playing Othello Using Monte Carlo.* Strategies, 2007: p. 1-9.
169. Kocsis, L., C. Szepesvári, and J. Willemson, *Improved monte-carlo search.* Univ. Tartu, Estonia, Tech. Rep, 2006. 1.
170. Kozelek, T., *Methods of MCTS and the game Arimaa.* Master's thesis, Charles University in Prague, 2009.
171. Sato, Y., D. Takahashi, and R. Grimbergen, *A shogi program based on Monte-Carlo tree search.* Icga Journal, 2010. 33(2): p. 80.
172. Romein, J.W. and H.E. Bal, *Solving awari with parallel retrograde analysis.* Computer, 2003. 36(10): p. 26-33.
173. Ramanujan, R. and B. Selman. *Trade-offs in sampling-based adversarial planning*. in *Proc. 21st Int. Conf. Automat. Plan. Sched., p. 202–209*. 2011. Freiburg, Germany
174. Nijssen, J. and M. Winands, *Enhancements for multi-player Monte-Carlo tree search.* In Proceedings of the 7th international conference on Computers and games, 2011: p. 238-249.
175. Finnsson, H., *Cadia-player: A general game playing agent*, 2007, Master's thesis, Reykjavik University, 200 7. http://www. ru. is/faculty/yngvi/hif-MSc-thesis. pdf.
176. Méhat, J. and T. Cazenave, *Combining uct and nested monte carlo search for single-player general game playing.* IEEE Transactions on Computational Intelligence and AI in Games, 2010. 2(4): p. 271-277.
177. Möller, M., M. Schneider, M. Wegner, and T. Schaub, *Centurio, a general game player: Parallel, Java-and ASP-based.* KI-Künstliche Intelligenz, 2011. 25(1): p. 17-24.

178. Rubin, J. and I. Watson, *Computer poker: A review.* Artificial intelligence, 2011. 175(5): p. 958-987.
179. Billings, D., A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron, *Game-tree search with adaptation in stochastic imperfect-information games.* In Proceedings of the 4th international conference on Computers and Games, 2006: p. 21-34.
180. Lanctot, M., K. Waugh, M. Zinkevich, and M. Bowling, *Monte Carlo sampling for regret minimization in extensive games.* Advances in neural information processing systems, 2009. 22: p. 1078-1086.
181. Johanson, M. and M. Bowling. *Data biased robust counter strategies*. in *Twelfth International Conference on Articial Intelligence and Statistics, p. 264-271*. 2009.
182. Johanson, M. *Calculating the Exploitability of Heads-Up Limit Agents*. in *2012 ACPC Symposium*. 2012.
183. van Lishout, F., G. Chaslot, and J.W.H.M. Uiterwijk. *Monte-Carlo tree search in backgammon*. in *Proc. Comput. Games Workshop, Amsterdam, Netherlands, p. 175–184*. 2007.
184. Baier, H. and M.H.M. Winands. *Beam Monte-Carlo Tree Search*. in *CIG 2012*. 2012. Granada, Spain.
185. Cazenave, T. *Reflexive monte-carlo search*. in *Computer Games Workshop 2007 (CGW 2007)*. 2007.
186. Robles, D. and S.M. Lucas. *A simple tree search method for playing Ms. Pac-Man*. in *Computer Intelligence and Games, CIG 2009*. 2009. IEEE.
187. Tanabe, Y., K. Yoshizoe, and H. Imai. *A study on security evaluation methodology for image-based biometrics authentication systems*. in *Proc. IEEE Conf. Biom.: Theory, Applicat. Sys.* 2009. Washington, DC, p. 1–6: IEEE.
188. Sabharwal, A., H. Samulowitz, and C. Reddy, *Guiding combinatorial optimization with UCT.* Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, 2012: p. 356-361.
189. Rimmel, A., F. Teytaud, and T. Cazenave, *Optimization of the nested Monte-Carlo algorithm on the traveling salesman problem with time windows.* Applications of Evolutionary Computation, 2011. Springer Berlin Heidelberg: p. 501-510.
190. Weinstein, A., C. Mansley, and M. Littman. *Sample-based planning for continuous action Markov Decision Processes*. in *Proc. 21st Int. Conf. Automat. Plan. Sched.* 2010. Freiburg, Germany, p. 335–338.
191. Coquelin, P.A. and R. Munos, *Bandit algorithms for tree search.* arXiv preprint cs/0703062, 2007.
192. Rimmel, A., *Improvements and Evaluation of the Monte Carlo Tree Search Algorithm*, 2009, Ph.D. dissertation, Lab. Rech. Inform. (LRI), Paris.
193. Satomi, B., Y. Joe, A. Iwasaki, and M. Yokoo. *Real-time solving of quantified CSPs based on Monte-Carlo game tree search*. in *Proc. 22nd Int. Joint Conf. Artif. Intell.* 2011. Barcelona, Spain, p. 655–662: AAAI Press.
194. Nakhost, H. and M. Müller. *Monte-Carlo exploration for deterministic planning*. in *Proc. 21st Int. Joint Conf. Artif. Intell.* 2009. Pasadena, California, p. 1766–1771.
195. Pellier, D., B. Bouzy, and M. Métivier, *An UCT Approach for Anytime Agent-Based Planning.* Advances in Practical Applications of Agents and Multiagent Systems, 2010: p. 211-220.
196. Chaslot, G., S. De Jong, J.T. Saito, and J. Uiterwijk. *Monte-Carlo tree search in production management problems*. in *Proc. BeNeLux Conf. Artif. Intell.* 2006. Namur, Belgium, p. 91–98.

197. Cazenave, T., F. Balbo, and S. Pinson. *Monte-Carlo bus regulation*. in *Proc. Int. IEEE Conf. Intell. Trans. Sys.* 2009. St Louis, Missouri, p. 340–345.
198. Tsai, H.J., S. Choudhry, M. Naqvi, W. Rodriguez-Cintron, E.G. Burchard, and E. Ziv, *Comparison of three methods to estimate genetic ancestry and control for stratification in genetic association studies among admixed populations.* Hum Genet, 2005. 118(3-4): p. 424-33.
199. Phillips, C., L. Prieto, M. Fondevila, A. Salas, A. Gomez-Tato, J. Alvarez-Dios, A. Alonso, A. Blanco-Verea, M. Brion, M. Montesino, A. Carracedo, and M.V. Lareu, *Ancestry analysis in the 11-M Madrid bomb attack investigation.* PLoS One, 2009. 4(8): p. e6583.