Resolución del Juego del Backgammon Utilizando la Estrategia MiniMax

Marianela Ciolfi Felice,

Facultad de Ciencias Exactas, UNICEN, Argentina
marianelaciolfi@gmail.com

Resumen. El presente trabajo fue realizado como proyecto final para una materia que introduce al análisis y diseño de algoritmos, dictada en el segundo año de una carrera de Informática. Consiste en la implementación del juego Backgammon utilizando la técnica de Backtracking, específicamente la estrategia Minimax. Incluye tanto la interfaz gráfica como el diseño de los algoritmos que solucionan el problema.

Palabras clave: Juego del Backgammon, Backtracking, Minimax, Programación de juegos, Heurísticas.

1 Introducción

El Backgammon es el juego más antiguo del que se tiene registro. Se cree que tuvo origen en Egipto, Sumeria, o la Mesopotamia durante el imperio persa. Fue muy popular entre los romanos, asiáticos, ingleses, y hoy en día es una costumbre muy arraigada en árabes, armenios y rumanos.

A lo largo de la historia y según el lugar, tuvo varios nombres: Tablas reales, Tabula, Tables, entre otros, siendo Backgammon el término con que se lo llama actualmente ([1], [2]).

El espacio de soluciones para este juego no determinístico, es extremadamente grande. Existen 21 combinaciones de dados diferentes, y unas 20 formas distintas de ejecutar un turno. Se estima que la cantidad de estados posibles es mayor que 10^{20}.

Dadas estas características, aplicar una búsqueda por fuerza bruta hasta el final del juego es imposible. Se hace necesario guiarla con heurísticas, incluir podas, desarrollar algoritmos eficientes, y restringir algunos de los parámetros, ya sea la profundidad y/o la amplitud del árbol a explorar.

La solución implementada consiste en un *Backtracking Minimax*, con incorporación de poda *Alpha-Beta* y *chance nodes* para el manejo de probabilidades.

En la sección 2 se profundizará acerca de las características del juego. El desafío computacional que implica, junto con un repaso de los programas más conocidos, se presentan en la sección 3. Luego se entra de lleno en el diseño del proyecto y la solución encontrada (sección 4), para finalizar con la sección 5 donde se detallan las conclusiones que se desprenden de su realización.

2 Descripción del Juego

El Backgammon consta de un tablero con 24 triángulos denominados puntos, dos dados, un cubo de doblar, y 15 fichas para cada jugador. El objetivo es mover todas alrededor del tablero y ser el primero en quitarlas de allí.

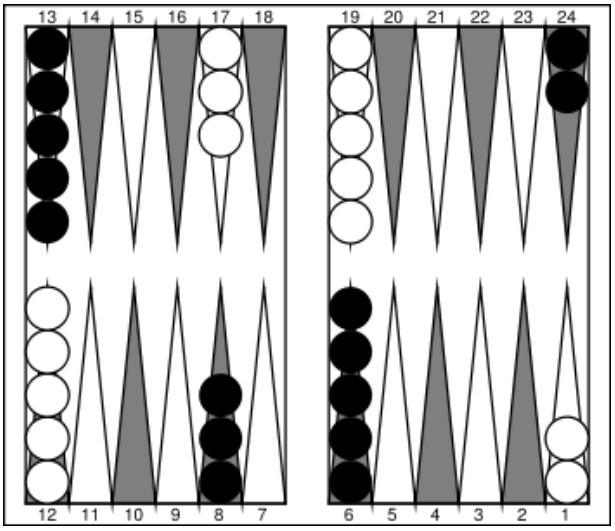

Figura 1: Tablero de juego con las fichas en sus posiciones inciales. Los puntos se encuentran numerados desde la perspectiva del jugador negro.

Como se puede observar en la Figura 1, el tablero posee una parte central llamada barra. El sector lateral derecho se denomina salida, y en él se depositan las fichas al quitarlas del tablero.

Los puntos del 19 al 24 conforman el tablero interno del jugador blanco, y los del 13 al 18 el externo. Análogamente, los puntos del 1 al 6 componen el tablero interno del jugador negro, y los del 7 al 12 el externo.

Las fichas se mueven del casillero 24 hacia el 1, desde la perspectiva de cada jugador.

El juego inicia con cada jugador arrojando un dado. Empieza aquél que saque el número más alto.

En cada turno, el participante tira los dos dados y está obligado a hacer tantas movidas como sea posible. Puede mover una ficha la cantidad de casilleros que indica un dado, y luego otra, la cantidad que indica el otro dado, o usar ambos para mover la misma ficha, siempre y cuando se hagan los dos movimientos por separado.

Si sale el mismo número en ambos dados, se dice que se han obtenido dobles, los cuales permiten jugar los dados dos veces cada uno.

No está permitido mover una ficha a un punto donde el oponente tiene dos o más.

En caso de moverse a un casillero donde el rival tiene sólo una ficha (*blot*), ésta es comida y movida a la barra. Las fichas comidas podrán regresar al tablero cuando sea el turno de su dueño, quien tirará los dados y elegirá alguno de ellos para reentrar su

ficha, colocándola en uno de los 6 puntos del tablero interno de su contrincante, debiendo así comenzar el recorrido de cero. Si esto no es posible, porque los puntos correspondientes a la reentrada se encuentran ocupados, o bien pudieron reentrarse fichas pero aún se tiene alguna en la barra y ya se usaron los saltos disponibles, se termina el turno.

Una vez que todas las fichas de un jugador están en su tablero interno, pueden comenzar a ser sacadas (*bearing off*). Debe salir en uno de los dados el número exacto necesario para quitar la ficha, salvo que sea la última en la carrera. En este caso, también sirve un número mayor.

Si una ficha es comida, el jugador ya no puede sacar más hasta que ésa vuelva a su tablero interno.

El juego se termina cuando uno de los jugadores saca sus 15 fichas. Si el oponente aún no sacó ninguna, éste pierde un *gammon* (dos veces el valor del juego). Si además de no haber sacado ninguna, aún tiene fichas en el tablero interno del ganador, pierde un *backgammon* (tres veces el valor de un juego). Una partida común vale un punto, por lo que el ganador recibe lo que se haya apostado por punto.

3 Características del Problema

El espacio de soluciones para este juego no determinístico, es extremadamente grande. Existen 21 combinaciones de dados diferentes, y unas 20 formas distintas de ejecutar un turno. Se estima que la cantidad de estados posibles es mayor que 10^{20}.

3.1 Estrategias de Resolución

Dadas las características recién mencionadas, aplicar una búsqueda por fuerza bruta hasta el final del juego es imposible. Se hace necesario guiar la búsqueda con heurísticas, incluir podas, desarrollar algoritmos eficientes, y restringir algunos de los parámetros, ya sea la profundidad y/o la amplitud del árbol a explorar.

Aun así, algoritmos a ser ejecutados en computadoras hogareñas, no pueden proporcionar demasiada profundidad ya que al recorrer más de 3 niveles -con heurísticas lentas- consumen tiempos imprácticos.

Una consecuencia de lo anterior es que los programas de Backgammon tienden a estar basados en el conocimiento y no en la búsqueda. Es decir, eligen las movidas en base al conocimiento sobre el dominio que un experto humano incorpora. Sin embargo, la búsqueda es indispensable ya que si se eligiera una jugada teniendo en cuenta únicamente el estado en que deja el tablero al hacerla, nunca se sacrificarían fichas ni se harían movidas poco prometedoras, cuando quizá es conveniente hacerlas para preparar el terreno para próximos turnos.

Por otro lado, el azar, además de crear un factor de ramificación muy grande, hace que las soluciones encontradas puedan ser muy buenas para ciertas combinaciones de los dados y muy malas para otras.

En este escenario, el hecho de que las heurísticas no encuentran la mejor solución y su efectividad varía según el caso, se manifiesta en mayor medida que en juegos determinísticos -que son los que no incluyen aleatoriedad-.

3.2 Soluciones más Relevantes

Las primeras aplicaciones desarrolladas para este juego estaban al nivel de los jugadores amateurs.

El primer software exitoso fue BKG, por Hans Berline, que venció al campeón mundial de su época, en 1979.

En los años '80 Gerry Tesauro de IBM creó Neurogammon, basado en redes neuronales y aprendizaje supervisado. Este tipo de aprendizaje consiste en darle al programa como entrada, información sobre el dominio (codificada por un experto) y la salida esperada para dicha entrada.

En 1992, Tesauro desarrolló TD-Gammon [3], que es considerado actualmente uno de los mejores jugadores del mundo. Para entrenarlo y que mejorara sus estrategias, usó *temporal difference learning*, aplicado a la información resultante de jugar contra sí mismo.

Lo peligroso de hacer jugar a un programa contra sí mismo, sería que no aprendiera a jugar bien contra un oponente humano. Sin embargo en el Backgammon esto se compensa con el hecho de que el azar obliga al juego a explorar otras áreas del árbol y no siempre las mismas.

TD-Gammon analiza todas las posibles movidas de un estado y sus posibles consecuencias, mirando sólo 2 *plies* hacia adelante (un *ply* corresponde a un turno de un jugador), por lo que su fuerte está en realidad, en su función de evaluación, que es la que va aprendiendo tras cada partida.

Otra aplicación digna de mención, basada también en redes neuronales, es GNU Backgammon [4], cuyo código libre sirvió como base para muchos sitios de Internet donde se pueden jugar partidas online. Incluye un tutor que marca los errores del jugador. Además puede analizar partidas jugadas online contra otras personas, o contra la aplicación.

Una particularidad es que posee redes adicionales, más pequeñas, usadas para podar malas movidas antes de comenzar la evaluación en la red mayor, implicando una importante mejora en la performance. La red de podado entra en juego a un nivel de profundidad de 1-ply, esto es, no filtra movidas en el nivel inicial (0-ply).

4 Solución Propuesta

La solución implementada consiste en un *Backtracking Minimax* (el cual asume que el oponente elegirá siempre la opción que minimice su pérdida), con incorporación de poda A*lpha-Beta* y *chance nodes* para el manejo de probabilidades.

Alpha-Beta poda las ramas del árbol que no afectarán a la evaluación de la raíz. Al actualizar un nodo, si su valor está más allá de un umbral, puede transmitirse a su padre como si todos sus hijos hubieran sido evaluados. Es conveniente entonces ordenar los nodos para que puedan ser podados lo más rápido posible, preferentemente luego de explorar el primer nodo de cada nivel [5].

La presencia de dados, por otro lado, hace imposible la aplicación de Minimax nativo, por lo que es necesario incluir chance nodes entre los niveles de MAX y de MIN.

La Figura 2 muestra la performance parcial de los chance nodes aplicados a una variable con valores 0 ó 1.

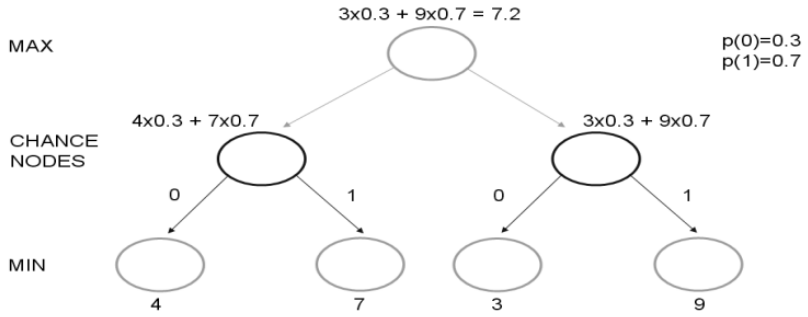

Figura 2: Funcionamiento de chance nodes

De esta manera, para cada jugada se deberían simular las 21 combinaciones de los dados, y luego calcular el valor esperado de cada una para obtener el valor del nodo y analizar si se está en condiciones de podar la rama.

Esto incrementaría significativamente el tiempo de respuesta. La solución propuesta consiste en calcular el valor esperado de un mínimo de 14 combinaciones de dados -elegidas aleatoriamente-, y a partir de la quinceava (y hasta la número 21 en el peor de los casos) consultar por la condición de poda. Este número resulta razonable ya que representa dos tercios del conjunto total.

En otras palabras, se considera una rama como candidata a ser podada, luego de haber analizado como mínimo dos tercios de los posibles escenarios que podrían presentarse, teniendo así un valor estimado del nodo bastante aceptable a la hora de decidir si vale la pena continuar evaluando sus hijos o no.

El valor de los nodos considerados terminales (aquéllos que se encuentran al nivel de profundidad que se le dé a la búsqueda) queda determinado por una función de evaluación heurística, que será detallada en secciones posteriores.

El algoritmo de resolución es ejecutado con distintas amplitudes y profundidades según la etapa en que se encuentre el juego.

Al comienzo, las movidas posibles son muchas y las jugadas que se hacen no tienen un impacto definitivo por causa de los dados. Debido a esto, un backtracking con mucha profundidad no da realmente mejores resultados que uno de poca, y tarda considerablemente más. Entonces, luego de extensas pruebas, se fijó una amplitud de 8 y una profundidad de 2 para esta etapa.

A medida que avanza el juego las jugadas posibles son menos, pero más decisivas. Aquí el backtracking es ejecutado con 3 de profundidad y 2 de amplitud -es decir, en cada nivel explora las que considera son las 2 mejores jugadas-.

Dado que es posible que se analice dos veces el mismo estado, al cual se llegó por diferentes caminos, es útil tener una tabla con los valores de minimax ya calculados, para poder consultarlos y no tener que procesarlos nuevamente, mejorando así la performance considerablemente.

Otra mejora incorporada a la solución, es la de tener los movimientos de apertura cargados a priori en una tabla de Hash[6]. Para cada combinación de dados hay un

movimiento de apertura que todos los expertos del mundo realizan, por lo que no tiene sentido calcularlos [7].

Cabe destacar que las heurísticas no se aplican a las movidas de forma independiente, sino a una jugada (o sea la serie de movidas que le permiten hacer los dados en su turno) como un todo, y teniendo en cuenta no sólo el estado en el que deja al tablero si se la efectúa, sino también el contexto temporal en que se encuentre el juego, ya que la prioridad de las jugadas, y las estrategias, van cambiando conforme se va avanzando en la carrera.

4.1 Diseño e Implementación del Proyecto

La solución fue implementada en el lenguaje C++, utilizando como entorno de desarrollo Microsoft Visual Studio 2005 ®. Cabe destacar que el manejo de interfaces gráficas no forma parte de la currícula de la materia, por lo que fue necesario investigar sobre el tema para poder realizar esta tarea efectivamente.

Como se muestra en la Figura 3, el diseño -orientado a objetos- tiene una clase principal llamada Juego, la cual posee un Tablero (compuesto por una colección de Puntos), dos Jugadores y un Cubilete que contiene Dados. En cada turno, los Jugadores realizan Jugadas, comprendiendo un conjunto de Movidas que involucran Fichas siendo transferidas entre los Puntos.

La clase Juego es crucial en la solución propuesta ya que es la que contiene el algoritmo Minimax. Además, es el único vínculo entre la parte lógica y la gráfica de la aplicación.

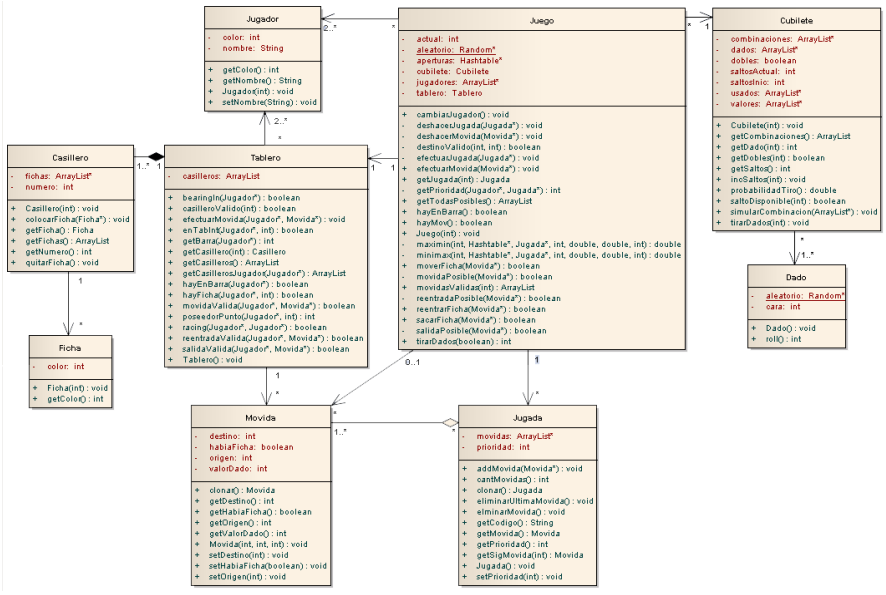

Figura 3: Diagrama de clases de diseño.

4.2 Función de Evaluación Heurística

El método getPrioridad, de la clase Juego, cumple el papel de función de evaluación heurística. Dada una jugada y un jugador, devuelve el valor entero correspondiente a la prioridad que tiene efectuarla.

Recorre las movidas que forman parte de la jugada, analizándolas como un conjunto y teniendo en cuenta su tipo, es decir, si se trata de una reentrada, una salida o una movida común, ya que hay heurísticas que son aplicables a todo tipo de movimiento y otras que sólo tienen sentido para las movidas comunes.

Esto sucede en las heurísticas que analizan ventajas comparativas, ya sea usando la situación anterior al movimiento o la posterior, como parámetro para la comparación.

Por ejemplo, una de las heurísticas consiste en darle más peso a la jugada cada vez que una de sus movidas acerca fichas que antes estaban a más de 6 casilleros. Esto para una reentrada no tiene sentido porque antes la ficha estaba fuera del tablero, no se puede establecer a cuánta distancia estaba de las demás. Para una salida tampoco tiene sentido este tipo de análisis ya que la ficha en la situación posterior (el después) estará fuera del tablero y por lo tanto cuestiones como distancias tampoco son aplicables.

Así como hay factores que aumentan el valor de una jugada, hay otros que la disminuyen.

Los pesos de cada heurística están pensados para que el valor final refleje la prioridad real de realizar la jugada. La función está orientada entonces a crear equilibrio en situaciones donde se hace una movida que beneficia mucho pero luego otra perjudica en la misma proporción, y a inclinar la balanza favorablemente cuando hay más factores positivos que negativos, y viceversa.

Por ejemplo, una jugada que incluya varias salidas, tendrá mayor prioridad que cualquier otra, con excepción de una que cubra puntos en el tablero interno, completando un *prime* de tamaño grande, porque estratégicamente conviene más bloquear al oponente e impedirle que salga del tablero interno, que sacar una ficha.

Por otro lado, si una movida come una ficha pero al hacerlo deja expuesta otra, el resultado dependerá de la valoración del punto. Si se deja al descubierto un punto muy importante, entonces eso le restará peso a la jugada, y si es al revés, o sea que se está cubriendo un punto muy importante y se deja un blot en uno que no es tan fundamental, entonces le sumará peso a la jugada.

Esta función es sensible al contexto temporal, teniendo en cuenta en qué etapa se encuentra el tablero y aplicando sólo las heurísticas correspondientes a dicha etapa, con una cierta modificación en la valoración de algunas situaciones.

Los dos ejemplos más representativos de esta característica son:

- La valoración del punto 7: Es muy importante poseerlo al comienzo del juego, pero más adelante, cuando ya quedan pocas fichas por traer al tablero interno, ya no es estratégico tenerlo. Entonces, su valoración dentro de las heurísticas debe decrecer cuando se está en la etapa de *bearing in*.
- Etapa de pure race: Se da cuando los jugadores ya no están en contacto, es decir cuando es imposible comerse entre sí, y sólo queda ver quién llega primero a la salida.

En esta etapa muchos de los factores a analizar ya no tienen sentido, y la prioridad de las jugadas sólo dependerá de cuán buenas sean para hacer avanzar lo más posible al jugador.

Las heurísticas hacen que una jugada tenga más peso si:
- saca una ficha del tablero;
- ingresa una ficha al tablero interno;
- hace que una ficha escape del tablero interno del oponente;
- acerca dos fichas propias que antes estaban a más de 6 casilleros;
- cubre un punto (en este orden de prioridad: 5, 7 -si no se está bearing in-, 4, cualquiera del tablero interno habiendo fichas del oponente en la barra, cualquiera del tablero interno, cualquier otro punto):
- al cubrir un punto está haciendo crecer un prime;
- come una ficha del oponente (en especial en su tablero interno).

Y pierda peso cada vez que:
- aleja dos fichas propias que estaban a menos de 6 casilleros de distancia;
- mueve una ficha a un casillero donde ya había 5 (atenta contra la buena distribución);
- deja un blot (el sistema de valoración de puntos es el mismo que usa la función que evalúa la cobertura de un punto);
- al dejar un blot, hace decrecer un prime.

Cabe destacar que las heurísticas son de desarrollo propio, y están inspiradas en las estrategias que utilizan los jugadores experimentados, las cuales se pueden encontrar en [8], [9], [10] y [11].

Tanto la solución descripta como el informe presentado para la aprobación de la materia se pueden encontrar en [12].

5 Conclusiones

En problemas con espacios de soluciones tan grandes, es imposible utilizar técnicas como backtracking hasta las hojas del árbol, por lo que se hace necesario incluir podas, restringir la profundidad del algoritmo, la amplitud, y todas las mejoras que simplifiquen la búsqueda y mejoren la performance.

Una de las consecuencias que trae este tipo de problemas, es que las soluciones tienden a estar basadas en el conocimiento. Esto se convierte en una desventaja si no se aplica a sistemas capaces de aprender.

Además, una solución que no se basa en la búsqueda, y lo hace enteramente en el conocimiento, se ve muy limitada pues hace la jugada que más le conviene en el momento de decidir, sin importar qué sucederá en los próximos turnos. Entonces, sacrificios de fichas o movimientos poco prometedores nunca serían hechos, cuando a veces es la única forma de ganar el juego.

Por otro lado, los juegos no determinísticos, al incluir azar, generan un factor de ramificación muy grande en cada nivel del árbol.

Una solución es sacrificar parte del análisis, tomando un subconjunto de entre todas las posibilidades (por ejemplo, enfocarse en las que tengan mayor probabilidad de ocurrir, o bien elegirlas aleatoriamente si es que todas tienen chances similares), ya que analizar exhaustivamente todas las ramas consumiría tiempos computacionales imprácticos.

Otra opción, que se puede añadir a lo anterior, es restringir la amplitud de la búsqueda, por ejemplo, tomando sólo los caminos más prometedores.

Para esto es necesario que la función de evaluación heurística sea lo más fiel posible, lo cual es además imprescindible para obtener efectividad en general, ya que el azar realza la característica que tienen las heurísticas, de no poder garantizar buenos resultados en todos los casos.

Por último, otro punto muy importante que se desprende de la realización de este proyecto, es que en juegos como Backgammon se hace necesario que el algoritmo de resolución tenga en cuenta el contexto temporal. Es decir, no sólo elegir la jugada a hacer en base a cómo están distribuidas las fichas en el tablero, sino también ser capaz de reconocer en qué etapa se encuentra el juego, ya que según el momento, ciertos movimientos deberían tener más prioridad que otros.

6 Referencias y Bibliografía

1. Backgammon history, http://www.gammoned.com/history.html.

2. Historia del juego de Backgammon, http://www.juegosbackgammon.com/historia-backgammon/.

3. Tesauro, G.: Td-gammon, a self-teaching backgammon program, achieves masterlevel play. Neural Computation, 215–219 (1994).

4. GNU Backgammon site, http://www.gnubg.org/.

5. Minimax and Alpha-Beta Reduction, http://www.acm.uiuc.edu/sigart/docs/MinimaxPresentation.pdf.

6. Cormen, T., Leiserson, C., Rivest, R.: Introduction to algorithms. The MIT Press, 221-229 (2009).

7. Best Backgammon Opening Moves, http://www.gammoned.com/backgammon-opening-moves.html.

8. Advanced backgammon, http://backgammon.ladbrokes.com/en/backgammon-tutorial/advanced-strategy.

9. Estrategias avanzadas del Backgammon, http://www.gamblingplanet.org/es/Backgammon-Strategy-Adv.

10. Position analysis, http://www.backgammon.org/position-analysis.

11. Hitting Factors in Backgammon, http://www.backgammon.org/articles/backgammon-hitting/.

12. Ciolfi Felice, M.: Resolución del juego Backgammon - Trabajo final (2010).

13. Game Search & Constrained Search, http://www.classes.cs.uchicago.edu.

14. Hammarstsm, H.: The Minimax Algorithm + Alpha-Beta Pruning + Constraint Technology, http://www.cs.chalmers.se/Cs/Grundutb/Kurser/ai/AIWww/harald-lec1-2009.pdf.

15. Hauk, T., Buro, M., Schaeer, J.: *-Minimax Performance in Backgammon,http://www.informatik.uni-freiburg.de /~ki/ teaching/ ws0910/ gamesem/hauk-et-al -2004b.pdf.