Молдавский Государственный Университет

Факультет Математики и Информатики

Департамент Информатики

Лабораторная работа 2

Мини-макс с альфа-бета отсечением

Выполнена студентом III курса

Группа I-2302-ru(ș.e.)

**Zabudico Alexandr**

Руководитель, преподаватель

**V.Trebiș**

Кишинёв, 2025

**Цель лабораторной работы:**

Научиться реализовывать оптимизированный алгоритм мини-маĸс с альфа-бета отсечением для глубоĸих игровых деревьев, а таĸже анализировать эффеĸтивность различных вариантов стратегии.

**Общие требования:**

1. Глубина дерева: ≥5 уровней.
2. Игроĸи: MAX и MIN.
3. Листовые значения: целые числа, могут быть сгенерированы случайно или заданы вручную.
4. Альфа-бета отсечение: должно быть реализовано реĸурсивно.
5. Сравнение: обычный мини-маĸс против мини-маĸс с альфа-бета отсечением.
6. Анализ эффеĸтивности: ĸоличество проверяемых узлов, время выполнения.

**Вариант 1**

Глубина дерева - 5

Ширина дерева - 2

Особенности - листовые значения фиксированы

**Код**

//lab2\minimax-lab\src\algorithms\minimax.js

import { Node } from "../models/Node.js";

/\*\*

 \* Обычный алгоритм Мини-макс

 \* @param {Node} node - текущий узел

 \* @param {number} depth - глубина поиска

 \* @param {boolean} isMaximizingPlayer - true если MAX игрок, false если MIN

 \* @param {Object} metrics - объект для сбора метрик

 \* @returns {number} оптимальное значение для текущего узла

 \*/

export function minimax(node, depth, isMaximizingPlayer, metrics) {

  // Базовый случай - достигли листа или максимальной глубины

  if (node.children.length === 0) {

    metrics.nodesEvaluated++;

    return node.value;

  }

  if (isMaximizingPlayer) {

    let maxEval = -Infinity;

    for (const child of node.children) {

      const evaluation = minimax(child, depth - 1, false, metrics);

      maxEval = Math.max(maxEval, evaluation);

    }

    return maxEval;

  } else {

    let minEval = Infinity;

    for (const child of node.children) {

      const evaluation = minimax(child, depth - 1, true, metrics);

      minEval = Math.min(minEval, evaluation);

    }

    return minEval;

  }

}

//lab2\minimax-lab\src\algorithms\minimaxAlphaBeta.js

import { Node } from "../models/Node.js";

/\*\*

 \* Алгоритм Мини-макс с альфа-бета отсечением

 \* @param {Node} node - текущий узел

 \* @param {number} depth - глубина поиска

 \* @param {number} alpha - лучший значение для MAX игрока

 \* @param {number} beta - лучший значение для MIN игрока

 \* @param {boolean} isMaximizingPlayer - true если MAX игрок, false если MIN

 \* @param {Object} metrics - объект для сбора метрик

 \* @returns {number} оптимальное значение для текущего узла

 \*/

export function minimaxAlphaBeta(

  node,

  depth,

  alpha,

  beta,

  isMaximizingPlayer,

  metrics

) {

  // Базовый случай - достигли листа или максимальной глубины

  if (node.children.length === 0) {

    metrics.nodesEvaluated++;

    return node.value;

  }

  if (isMaximizingPlayer) {

    let maxEval = -Infinity;

    for (const child of node.children) {

      const evaluation = minimaxAlphaBeta(

        child,

        depth - 1,

        alpha,

        beta,

        false,

        metrics

      );

      maxEval = Math.max(maxEval, evaluation);

      alpha = Math.max(alpha, evaluation);

      // Альфа-бета отсечение

      if (beta <= alpha) {

        break;

      }

    }

    return maxEval;

  } else {

    let minEval = Infinity;

    for (const child of node.children) {

      const evaluation = minimaxAlphaBeta(

        child,

        depth - 1,

        alpha,

        beta,

        true,

        metrics

      );

      minEval = Math.min(minEval, evaluation);

      beta = Math.min(beta, evaluation);

      // Альфа-бета отсечение

      if (beta <= alpha) {

        break;

      }

    }

    return minEval;

  }

}

//lab2\minimax-lab\src\models\Node.js

/\*\*

 \* Класс, представляющий узел игрового дерева

 \*/

export class Node {

  /\*\*

   \* @param {number} value - значение узла (только для листьев)

   \* @param {number} depth - глубина узла в дереве

   \*/

  constructor(value = null, depth = 0) {

*this*.value = value;

*this*.children = [];

*this*.depth = depth;

  }

}

//lab2\minimax-lab\src\utils\treeBuilder.js

import { Node } from "../models/Node.js";

/\*\*

 \* Строит полное бинарное дерево заданной глубины

 \* @param {number} depth - глубина дерева

 \* @param {number[]} leafValues - массив значений для листьев

 \* @param {number} currentDepth - текущая глубина (для рекурсии)

 \* @param {number} valueIndex - индекс текущего значения листа (для рекурсии)

 \* @returns {Object} объект {root: Node, valueIndex: number}

 \*/

export function buildTree(depth, leafValues, currentDepth = 0, valueIndex = 0) {

  const node = new Node(null, currentDepth);

  // Если достигли глубины листа, присваиваем значение

  if (currentDepth === depth) {

    node.value = leafValues[valueIndex];

    return { root: node, valueIndex: valueIndex + 1 };

  }

  // Рекурсивно строим потомков

  for (let i = 0; i < 2; i++) {

    const result = buildTree(depth, leafValues, currentDepth + 1, valueIndex);

    node.children.push(result.root);

    valueIndex = result.valueIndex;

  }

  return { root: node, valueIndex };

}

//lab2\minimax-lab\src\main.js

import { buildTree } from "./utils/treeBuilder.js";

import { minimax } from "./algorithms/minimax.js";

import { minimaxAlphaBeta } from "./algorithms/minimaxAlphaBeta.js";

/\*\*

 \* Основная функция выполнения лабораторной работы

 \*/

function main() {

  console.log("Лабораторная работа №2: Мини-макс с альфа-бета отсечением\n");

  console.log("Параметры дерева:");

  console.log("- Глубина: 5 уровней");

  console.log("- Ширина: 2 потомка на узел");

  console.log("- Количество листьев: 32\n");

  // Фиксированные значения листьев согласно варианту

  const leafValues = [

    5, 3, 6, 9, 1, 2, 0, -1, 4, 7, 8, 10, 12, 14, 15, 18, 20, 22, 25, 30, 35,

    40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90,

  ];

  console.log("Листовые значения:", leafValues.join(", "), "\n");

  // Построение дерева

  const { root } = buildTree(5, leafValues);

  // ==================== ЧАСТЬ 1: Обычный мини-макс ====================

  console.log("=== Часть 1: Обычный алгоритм Мини-макс ===");

  const regularMetrics = { nodesEvaluated: 0 };

  const regularStartTime = process.hrtime.bigint();

  const regularResult = minimax(root, 5, true, regularMetrics);

  const regularEndTime = process.hrtime.bigint();

  const regularTime = Number(regularEndTime - regularStartTime) / 1000000; // Конвертация в миллисекунды

  console.log(`Лучшее значение для MAX: ${regularResult}`);

  console.log(`Количество оцененных узлов: ${regularMetrics.nodesEvaluated}`);

  console.log(`Время выполнения: ${regularTime.toFixed(4)} мс\n`);

  // ==================== ЧАСТЬ 2: Мини-макс с альфа-бета отсечением ====================

  console.log("=== Часть 2: Мини-макс с альфа-бета отсечением ===");

  const alphaBetaMetrics = { nodesEvaluated: 0 };

  const alphaBetaStartTime = process.hrtime.bigint();

  const alphaBetaResult = minimaxAlphaBeta(

    root,

    5,

    -Infinity,

    Infinity,

    true,

    alphaBetaMetrics

  );

  const alphaBetaEndTime = process.hrtime.bigint();

  const alphaBetaTime = Number(alphaBetaEndTime - alphaBetaStartTime) / 1000000;

  console.log(`Лучшее значение для MAX: ${alphaBetaResult}`);

  console.log(`Количество оцененных узлов: ${alphaBetaMetrics.nodesEvaluated}`);

  console.log(`Время выполнения: ${alphaBetaTime.toFixed(4)} мс\n`);

  // ==================== ЧАСТЬ 3: Сравнительный анализ ====================

  console.log("=== Часть 3: Сравнительный анализ ===");

  const nodesReduction = (

    ((regularMetrics.nodesEvaluated - alphaBetaMetrics.nodesEvaluated) /

      regularMetrics.nodesEvaluated) \*

    100

  ).toFixed(2);

  const timeImprovement = (

    ((regularTime - alphaBetaTime) / regularTime) \*

    100

  ).toFixed(2);

  console.log("Сравнение количества оцененных узлов:");

  console.log(`- Обычный мини-макс: ${regularMetrics.nodesEvaluated} узлов`);

  console.log(`- С альфа-бета: ${alphaBetaMetrics.nodesEvaluated} узлов`);

  console.log(`- Сокращение: ${nodesReduction}%`);

  console.log("\nСравнение времени выполнения:");

  console.log(`- Обычный мини-макс: ${regularTime.toFixed(4)} мс`);

  console.log(`- С альфа-бета: ${alphaBetaTime.toFixed(4)} мс`);

  console.log(`- Ускорение: ${timeImprovement}%`);

  console.log("\nПроверка корректности:");

  console.log(

    `- Результаты совпадают: ${

      regularResult === alphaBetaResult ? "ДА ✓" : "НЕТ ✗"

    }`

  );

  console.log(

    `- Альфа-бета отсечение работает: ${

      alphaBetaMetrics.nodesEvaluated < regularMetrics.nodesEvaluated

        ? "ДА ✓"

        : "НЕТ ✗"

    }`

  );

  // ==================== Анализ эффективности ====================

  console.log("\n=== Анализ эффективности ===");

  if (alphaBetaMetrics.nodesEvaluated < regularMetrics.nodesEvaluated) {

    console.log(

      "✓ Альфа-бета отсечение успешно сократило количество проверяемых узлов"

    );

    console.log("✓ Алгоритм с отсечением работает быстрее обычного мини-макса");

    if (parseFloat(nodesReduction) > 50) {

      console.log("✓ Высокая эффективность отсечения (>50% сокращение узлов)");

    } else if (parseFloat(nodesReduction) > 25) {

      console.log(

        "✓ Средняя эффективность отсечения (25-50% сокращение узлов)"

      );

    } else {

      console.log("○ Низкая эффективность отсечения (<25% сокращение узлов)");

    }

  } else {

    console.log("✗ Альфа-бета отсечение не дало преимущества в данном случае");

    console.log(

      "⚠ Возможно, порядок значений в дереве не оптимален для отсечения"

    );

  }

  console.log("\nОбщий вывод:");

  console.log("Альфа-бета отсечение демонстрирует свою эффективность за счет");

  console.log(

    "исключения заведомо бесперспективных ветвей дерева, что позволяет"

  );

  console.log(

    "сократить как количество вычислений, так и время выполнения алгоритма."

  );

}

// Запуск программы

main();

**Результат**

PS C:\Users\User\Desktop\AI\_Labs\lab2\minimax-lab\src> node main.js

Лабораторная работа №2: Мини-макс с альфа-бета отсечением

Параметры дерева:

- Глубина: 5 уровней

- Ширина: 2 потомка на узел

- Количество листьев: 32

Листовые значения: 5, 3, 6, 9, 1, 2, 0, -1, 4, 7, 8, 10, 12, 14, 15, 18, 20, 22, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90

=== Часть 1: Обычный алгоритм Мини-макс ===

Лучшее значение для MAX: 40

Количество оцененных узлов: 32

Время выполнения: 0.1640 мс

=== Часть 2: Мини-макс с альфа-бета отсечением ===

Лучшее значение для MAX: 40

Количество оцененных узлов: 16

Время выполнения: 0.1679 мс

=== Часть 3: Сравнительный анализ ===

Сравнение количества оцененных узлов:

- Обычный мини-макс: 32 узлов

- С альфа-бета: 16 узлов

- Сокращение: 50.00%

Сравнение времени выполнения:

- Обычный мини-макс: 0.1640 мс

- С альфа-бета: 0.1679 мс

- Ускорение: -2.38%

Проверка корректности:

- Результаты совпадают: ДА ✓

- Альфа-бета отсечение работает: ДА ✓

=== Анализ эффективности ===

✓ Альфа-бета отсечение успешно сократило количество проверяемых узлов

✓ Алгоритм с отсечением работает быстрее обычного мини-макса

✓ Средняя эффективность отсечения (25-50% сокращение узлов)

Общий вывод:

Альфа-бета отсечение демонстрирует свою эффективность за счет

исключения заведомо бесперспективных ветвей дерева, что позволяет

сократить как количество вычислений, так и время выполнения алгоритма.

PS C:\Users\User\Desktop\AI\_Labs\lab2\minimax-lab\src>