Пермский Национальный Исследовательский Политехнический Университет
Электротехнический Факультет
Кафедра ИТАС

Курсовая работа

По дисциплине “Дискретная математика”

Применение алгоритма Магу к задачам о
расстановке шахматных фигур.

Выполнил: Денисов А., гр. ЭВТ-11
Проверил: Соловьев А. Е.

Пермь, 2012

Постановка задачи

На шахматной доске размером 4x4 найти такие расстановки с максимальным числом ферзей, чтобы они не били друг друга т. е. найти множества внутренней устойчивости в неориентированном графе G=<V,E>, где множество вершин V образовано клетками шахматной доски, а множество граней E - возможными атаками фигуры.

Для поиска множеств внутренней устойчивости используется алгоритм Магу.

Алгоритм Магу для поиска мн-в внутренней устойчивости

1. Построить матрицу смежности для графа.
2. По единицам матрицы построить парные дизъюнкты.
3. Преобразовать полученное выражение к ДНФ, выполнив все упрощения.
4. Для каждой конъюнкции в ДНФ получить недостающие вершины.

Полученные на 4 шаге множества дополнений являются множествами внутренней устойчивости. Чтобы из этих множеств получить максимальные расстановки фигур, нужно выбрать только множества с максимальной мощностью (число внутренней устойчивости).

Решение

Для решения задачи была написана программа на языке программирования C++, реализующая шаги алгоритма.

Код программы

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>

using namespace std;

//Дизъюнкция двух элементов (x ∨ y)
struct xory
{
    xory(unsigned _x,unsigned _y) : x(_x),y(_y) {}
    unsigned short x;
    unsigned short y;
};

typedef vector<vector<short> > DNFexpr;
typedef vector<xory> CNFexpr;

//Позиция на шахматной доске
struct chess_pos
{
    chess_pos(unsigned short num, unsigned size)
    {
        /* Преобразовать номер позиции (@num)
         * в шахматное представление на доске размером @size
         * Пример: chess_pos(5,3) = b2; chess_pos(16,4) = d1
         */
        r = '1' + (size - (num / size)) - 1;
        c = 'a' + (num % size);
    }
    //Печать позиции
    friend ostream& operator<<(ostream& out,chess_pos p) {
        out << p.c << p.r; return out;
    }

    char r;
    char c;
};
```

```
//Матрица смежности
class Matrix : public vector<vector<bool> >
{
public:
    /* Создать матрицу смежности
     * Парметры:
     * @func(x,y,x0,y0): функция, проверяющая находится ли клетка под ударом
     *     Принимает клетку для проверки (x,y) и положение фигуры (x0,y0)
     * @board_size: размер шахматной доски
     */
    Matrix(bool (&func)(int,int,int,int),unsigned board_size) :
    vector<vector<bool> >
    (board_size*board_size,vector<bool>(board_size*board_size))
    {
        /* Для каждого возможного положения фигуры (piece_x,piece_y)
         * вычислить клетки под боем (x,y).
         */
        for(int piece_x = 0; piece_x < board_size; piece_x++)
        for(int piece_y = 0; piece_y < board_size; piece_y++)
            for(int y = 0; y < board_size; y++)
            for(int x = 0; x < board_size; x++)
                this->at(piece_y*board_size+piece_x).at(y*board_size+x) = \
                    func(x,y,piece_x,piece_y);

        this->board_size = board_size;
    }
    //Печать матрицы
    friend ostream& operator<<(ostream &stream, const Matrix &m)
    {
        stream << "   ";
        for(int i=0; i < m.size();i++)
            stream << chess_pos(i,m.board_size).c
                   << chess_pos(i,m.board_size).r << ' ';
        stream << endl;

        for(int row = 0; row < m.size(); row++)
        {
            stream << chess_pos(row,m.board_size).c
                   << chess_pos(row,m.board_size).r << ' ';
            for(int col = 0; col < m[row].size(); col++)
                stream << m[row][col] << "  ";
            stream << endl;
        }
        return stream;
    }
private:
    unsigned board_size;
};
```

```
// Класс для работы с логическими высказываниями
class BoolExpr
{
    public:
        /* Инициализировать класс используя матрицу смежности (создать КНФ)
         * Параметры:
         * @m - матрица смежности
         * @board_size - размер доски
         */
        BoolExpr(Matrix& m,unsigned board_size) : is_dnf(false)
        {
            expr = new CNFexpr;
            dnf = NULL;
            /* Добавить в выражение дизьюнкции для каждой еденицы в матрице
             * (с учетом симметричности матрицы)*/
            for(int r = 0; r < m.size(); r++)
                for(int c = 0; c < m.size(); c++)
                    if((m[r][c] == true) && (r < c))
                        expr->push_back(xory(r,c));

            this->board_size = board_size;
        }
        /* Инициализировать класс используя готовое выражение
         * Параметры:
         * @_dnf: выражение в ДНФ
         */
        BoolExpr(DNFexpr* _dnf,unsigned board_size) : is_dnf(true)
        {
            expr = NULL; dnf = _dnf;
            this->board_size = board_size;
        }

        ~BoolExpr() { if(is_dnf) delete dnf; delete expr; }

        /* Преобразовать выражение в ДНФ */
        void toDNF(void (*process)(unsigned,unsigned)=NULL) {
            if(is_dnf) return;
            dnf = new DNFexpr();

            /* Перемножить каждую скобку в исходном выражении (@expr)
             * и выполнить поглощения. Результат записывается в @dnf */
            for(int i = 0; i < expr->size(); i++) {
                //Перемножение очередной скобки
                Multiply(expr->at(i),*dnf);
                Simplify(dnf);
                if(process) (*process)(i,expr->size());
            }
            delete expr;
            is_dnf = true;
        }
```

```
/* Получить дополнения каждой элементарной коньюнкции в ДНФ
 * Пример: для переменных a,b,c,d,e:
 * getComplement(ac or be or abd) = bde or acd or ce
 */
BoolExpr* getComplement()
{
    if(!is_dnf) return NULL;

    DNFexpr *cmpl = new DNFexpr();

    for(DNFexpr::iterator it = dnf->begin(); it < dnf->end(); it++)
    {
        vector<short> inv;
        for(int i = 0; i < board_size*board_size; i++)
            if(!Contains(vector<short>(1,i),*it))
                inv.push_back(i);
        cmpl->push_back(inv);
    }

    return new BoolExpr(cmpl,this->board_size);
}

/* Получить все конъюнкции с максимальным числом элементов в ДНФ */
DNFexpr* getMaxConjs()
{
    if(!isDNF()) return NULL;

    DNFexpr *c = new DNFexpr();

    int n = getMaxConjSize();
    for(int i = 0; i < dnf->size(); i++)
        if(dnf->at(i).size() == n)
            c->push_back(dnf->at(i));

    return c;
}

/* Получить размер коньюнкции с максимальным числом элементов в ДНФ*/
int getMaxConjSize()
{
    if(!isDNF())
        return -1;
    unsigned max = dnf->at(0).size();
    for(int i =1; i < dnf->size(); i++)
    {
        if(dnf->at(i).size() > max)
            max = dnf->at(i).size();
    }
    return max;
}
```

```
    /* Является ли выражение ДНФ */
    bool isDNF() { return this->is_dnf; }

    //Печать выражения
    friend ostream& operator<<(ostream &stream, BoolExpr &m) {
        if(m.isDNF()) m.PrintDNF(stream);
        else m.PrintExpr(stream);
        return stream;
    }

private:
    CNFexpr* expr; //КНФ
    DNFexpr* dnf; //ДНФ
    bool is_dnf; //форма выражения
    unsigned board_size;

    /* Является ли @subset подмножеством @set */
    bool Contains(const vector<short> &subset,const vector<short> &set)
    {
        if(!subset.size()) return false;
        for(int i = 0; i < subset.size(); i++) {
            bool has = false;
            for(int j = 0; j < set.size(); j++) {
                if(set[j] == subset[i]) {
                    has = true;
                    break;
                }
            }
            if(!has)
                return false;
        }

        return true;
    }

    /* Выполнить поглощения в ДНФ (@what)*/
    void Simplify(DNFexpr* what)
    {
        for(DNFexpr::iterator it=what->begin(); it < what->end(); it++)
        for(DNFexpr::iterator it2=what->begin(); it2 < what->end();it2++)
            if((it!=it2) && Contains(*it,*it2))
                 //удалить элем. конъюнкцию @it2 если она содержит @it
                (*it2).clear();

        for(int i = 0; i < what->size(); i++)
            if(what->at(i).size() == 0) {
                what->erase(what->begin()+i);
                i--;
            }
    }
```

```
/* Раскрыть конъюнкцию дизъюнкции двух элементов (@what)
 * и выражения в ДНФ (@v)
 * Параметры:
 * @what: дизъюнкция двух элементов (x or y)
 * @v: выражение
 * Результат записывается в @v
 * Пример:
 * Multiply((a or b),(bc or ab or cd))=
 * abc or ab or acd or bc or ab or bcd
 */
void Multiply(xory what,DNFexpr &v)
{
    //умножение @what на пустую скобку
    if(v.size() == 0)
    {
        vector<short> a1(1,what.x);
        vector<short> b1(1,what.y);
        v.push_back(a1);
        v.push_back(b1);
        return;
    }

    DNFexpr mul_b;

    /* Для каждой элементарной конъюнкции в @v */
    for(DNFexpr::iterator it = v.begin(); it < v.end(); it++)
    {
        /* Содержит ли текущая конъюнкцию x или y
         * (возможно ли применить идемпотентность) */
        bool found_x = Contains(vector<short>(1,what.x),*it);
        bool found_y = Contains(vector<short>(1,what.y),*it);

        //результат умножения @v на @what.y
        vector<short> mb(*it);

        //умножаем на y
        if(!found_y)
            mb.push_back(what.y);

        mul_b.push_back(mb);

        //умножаем на x
        if(!found_x)
            it->push_back(what.x);
    }
    //Добавляем в @v результат умножения на @what.y
    for(DNFexpr::iterator it = mul_b.begin(); it < mul_b.end(); it++)
        v.push_back(*it);
}
```

```
        //Печать выражения (если оно в ДНФ)
        void PrintDNF(ostream& out)
        {
            const string or_str = " ∨ ";//"or";
            const string and_str = " ∧ ";// "and";

            for(DNFexpr::iterator it = dnf->begin(); it < dnf->end(); it++)
            {
                out << "(";
                for(int j = 0; j < it->size(); j++)
                    out << chess_pos(it->at(j),this->board_size)
                        << ((j == it->size()-1) ? "" : and_str);
                out << ")";
                out << ((it==dnf->end()-1) ? "" : or_str) << endl;
            }
        }

        //Печать выражения (если оно в КНФ)
        void PrintExpr(ostream& out,unsigned wrap=3)
        {
            const string or_str = " ∨ ";//"or";
            const string and_str =" ∧ "; //"and";

            int items = wrap;
            for(int i =0; i < expr->size(); i++)
            {
                chess_pos xpos(expr->at(i).x,this->board_size);
                chess_pos ypos(expr->at(i).y,this->board_size);

                out << "(" << xpos << or_str << ypos
                    << ") " + ((i == expr->size()-1) ? "" : and_str);

                if(items == 0) {
                    items = wrap;
                    out << endl;
                }
                else
                    items--;
            }
            out << endl;
        }
};
```

```
bool queen(int x,int y,int x0,int y0)
{
    return ((x == x0) || //горизонтали
           (y == y0) || //вертикли
           //диагонали (arctg(|dx/dy|)=45)
           (abs(float(x0-x)/float(y0-y)) == 1.0f)) &&
           !((x==x0) && (y==y0));
}

int main(int argc,char** argv)
{
    int size = 4;
    cout << "Размер доски?\n>"; cin >> size;

    ostream *fout;
    string f;
    cout << "Файл? (\"cout\" - на экран)\n>";
    cin >> f;
    if(f == "cout") fout = &cout;
    else fout = new ofstream(f.c_str());

    *fout << "Матрица смежности:" << endl;
    Matrix m(queen,size);
    *fout << m << endl;

    BoolExpr expr(m,size);
    *fout << "Условие несовместимости:\n" << expr;
    expr.toDNF();
    *fout << "ДНФ:\n";
    *fout << expr << endl;
    *fout << "Дополнения конъюнкций ДНФ:\n";
    BoolExpr* cmpl = expr.getComplement();
    *fout << *cmpl << endl;
    *fout << "Число внутренней устойчивости:\n";
    *fout << cmpl->getMaxConjSize() << endl;
    *fout << "Расстановки с максимальным числом фигур" << endl;
    DNFexpr* st = cmpl->getMaxConjs();

    for(int i = 0; i < st->size(); i++)
    {
        for(int j = 0; j < (*st)[i].size(); j++)
        {
            *fout << chess_pos(st->at(i).at(j),size) << " ";
        }
        *fout << endl;
    }

    return 0;
}
```

Результаты

Результаты работы программы представлены ниже (сгруппированы по шагам алгоритма Магу).

Матрица смежности

Шаг 1: Построение матрицы смежности

	a4	b4	c4	d4	a3	b3	c3	d3	a2	b2	c2	d2	a1	b1	c1	d1
a4	0	1	1	1	1	1	0	0	1	0	1	0	1	0	0	1
b4	1	0	1	1	1	1	1	0	0	1	0	1	0	1	0	0
c4	1	1	0	1	0	1	1	1	1	0	1	0	0	0	1	0
d4	1	1	1	0	0	0	1	1	0	1	0	1	1	0	0	1
a3	1	1	0	0	0	1	1	1	1	1	0	0	1	0	1	0
b3	1	1	1	0	1	0	1	1	1	1	1	0	0	1	0	1
c3	0	1	1	1	1	1	0	1	0	1	1	1	1	0	1	0
d3	0	0	1	1	1	1	1	0	0	0	1	1	0	1	0	1
a2	1	0	1	0	1	1	0	0	0	1	1	1	1	1	0	0
b2	0	1	0	1	1	1	1	0	1	0	1	1	1	1	1	0
c2	1	0	1	0	0	1	1	1	1	1	0	1	0	1	1	1
d2	0	1	0	1	0	0	1	1	1	1	1	0	0	0	1	1
a1	1	0	0	1	1	0	1	0	1	1	0	0	0	1	1	1
b1	0	1	0	0	0	1	0	1	1	1	1	0	1	0	1	1
c1	0	0	1	0	1	0	1	0	0	1	1	1	1	1	0	1
d1	1	0	0	1	0	1	0	1	0	0	1	1	1	1	1	0

Условие несовместимости

Шаг 2: Построение условия несовместимости (с учетом симметричности матрицы)

$$(a4 \vee b4) \wedge (a4 \vee c4) \wedge (a4 \vee d4) \wedge (a4 \vee a3) \wedge (a4 \vee b3) \wedge (a4 \vee a2) \wedge (a4 \vee c2) \wedge (a4 \vee a1) \wedge (a4 \vee d1) \wedge (b4 \vee c4) \wedge (b4 \vee d4) \wedge (b4 \vee a3) \wedge (b4 \vee b3) \wedge (b4 \vee c3) \wedge (b4 \vee b2) \wedge (b4 \vee d2) \wedge (b4 \vee b1) \wedge (c4 \vee d4) \wedge (c4 \vee b3) \wedge (c4 \vee c3) \wedge (c4 \vee d3) \wedge (c4 \vee a2) \wedge (c4 \vee c2) \wedge (c4 \vee c1) \wedge (d4 \vee c3) \wedge (d4 \vee d3) \wedge (d4 \vee b2) \wedge (d4 \vee d2) \wedge (d4 \vee a1) \wedge (d4 \vee d1) \wedge (a3 \vee b3) \wedge (a3 \vee c3) \wedge (a3 \vee d3) \wedge (a3 \vee a2) \wedge (a3 \vee b2) \wedge (a3 \vee a1) \wedge (a3 \vee c1) \wedge (b3 \vee c3) \wedge (b3 \vee d3) \wedge (b3 \vee a2) \wedge (b3 \vee b2) \wedge (b3 \vee c2) \wedge (b3 \vee b1) \wedge (b3 \vee d1) \wedge (c3 \vee d3) \wedge (c3 \vee b2) \wedge (c3 \vee c2) \wedge (c3 \vee d2) \wedge (c3 \vee a1) \wedge (c3 \vee c1) \wedge (d3 \vee c2) \wedge (d3 \vee d2) \wedge (d3 \vee b1) \wedge (d3 \vee d1) \wedge (a2 \vee b2) \wedge (a2 \vee c2) \wedge (a2 \vee d2) \wedge (a2 \vee a1) \wedge (a2 \vee b1) \wedge (b2 \vee c2) \wedge (b2 \vee d2) \wedge (b2 \vee a1) \wedge (b2 \vee b1) \wedge (b2 \vee c1) \wedge (c2 \vee d2) \wedge (c2 \vee b1) \wedge (c2 \vee c1) \wedge (c2 \vee d1) \wedge (d2 \vee c1) \wedge (d2 \vee d1) \wedge (a1 \vee b1) \wedge (a1 \vee c1) \wedge (a1 \vee d1) \wedge (b1 \vee c1) \wedge (b1 \vee d1) \wedge (c1 \vee d1)$$

ДНФ

Шаг 3: Преобразование выражения к ДНФ

(a4 ∧ b4 ∧ d4 ∧ b3 ∧ c3 ∧ d3 ∧ a2 ∧ c2 ∧ c1 ∧ b2 ∧ a1 ∧ d2 ∧ b1) ∨
(a4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ c3 ∧ b2 ∧ d2 ∧ b1 ∧ d3 ∧ c2 ∧ a1 ∧ c1) ∨
(a4 ∧ b4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ d3 ∧ b2 ∧ c2 ∧ d2 ∧ a1 ∧ c1 ∧ b1) ∨
(a4 ∧ b4 ∧ d4 ∧ b3 ∧ c3 ∧ d3 ∧ a2 ∧ c2 ∧ c1 ∧ a3 ∧ d2 ∧ a1 ∧ b1) ∨
(b4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ a2 ∧ c2 ∧ a1 ∧ d1 ∧ c3 ∧ d2 ∧ b1 ∧ c1) ∨
(a4 ∧ b4 ∧ c4 ∧ c3 ∧ d3 ∧ b2 ∧ d2 ∧ a1 ∧ d1 ∧ b3 ∧ a2 ∧ c1 ∧ b1) ∨
(a4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ c3 ∧ b2 ∧ d2 ∧ b1 ∧ c2 ∧ d1 ∧ a2 ∧ c1) ∨
(a4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ c3 ∧ b2 ∧ d2 ∧ b1 ∧ d3 ∧ a2 ∧ c1 ∧ d1) ∨
(a4 ∧ b4 ∧ d4 ∧ b3 ∧ c3 ∧ d3 ∧ a2 ∧ c2 ∧ c1 ∧ a3 ∧ b2 ∧ d1 ∧ b1) ∨
(a4 ∧ b4 ∧ c4 ∧ d4 ∧ a3 ∧ c3 ∧ d3 ∧ a2 ∧ b2 ∧ c2 ∧ b1 ∧ d1 ∧ c1) ∨
(a4 ∧ b4 ∧ c4 ∧ c3 ∧ d3 ∧ b2 ∧ d2 ∧ a1 ∧ d1 ∧ b3 ∧ a2 ∧ c1 ∧ c2) ∨
(b4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ a2 ∧ c2 ∧ a1 ∧ d1 ∧ d3 ∧ b2 ∧ d2 ∧ c1) ∨
(b4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ a2 ∧ c2 ∧ a1 ∧ d1 ∧ c3 ∧ d3 ∧ b2 ∧ c1) ∨
(a4 ∧ b4 ∧ c4 ∧ c3 ∧ d3 ∧ b2 ∧ d2 ∧ a1 ∧ d1 ∧ a3 ∧ a2 ∧ c2 ∧ b1) ∨
(a4 ∧ b4 ∧ c4 ∧ c3 ∧ d3 ∧ b2 ∧ d2 ∧ a1 ∧ d1 ∧ a3 ∧ b3 ∧ c2 ∧ b1) ∨
(b4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ a2 ∧ c2 ∧ a1 ∧ d1 ∧ c3 ∧ d2 ∧ b1 ∧ b2) ∨
(a4 ∧ c4 ∧ d4 ∧ a3 ∧ b3 ∧ c3 ∧ b2 ∧ d2 ∧ b1 ∧ c2 ∧ d1 ∧ a1) ∨
(a4 ∧ b4 ∧ d4 ∧ b3 ∧ c3 ∧ d3 ∧ a2 ∧ c2 ∧ c1 ∧ b2 ∧ a1 ∧ d1)

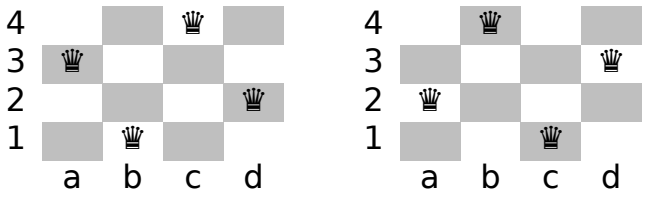
Множества внутренней устойчивости

Шаг 4: Получение дополнений элементарных конъюнкций в ДНФ

{c4, a3, d1}, {b4, a2, d1}, {c3, a2, d1}, {c4, b2, d1}
{a4, d3, b2}, {d4, a3, c2}, {b4, d3, a1}, {b4, c2, a1}
{c4, d2, a1}, {b3, d2, a1}, {d4, a3, b1}, {a4, c3, b1}
{a4, d2, b1}, {d4, b3, c1}, {d4, a2, c1}, {a4, d3, c1}
{c4, a3, d2, b1}, {b4, d3, a2, c1}

Здесь множества с наибольшим числом элементов {c4, a3, d2, b1} и {b4, d3, a2, c1} и являются максимальными расстановками ферзей. Количество элементов в этих множествах (число внутренней устойчивости) равно четырем.

Расстановки на шахматной доске:



Вывод

Таким образом, на шахматной доске размером 4x4 возможны две расстановки с максимальным числом взаимно неатакующих ферзей, равным четырем.