ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TỐI ƯU LẬP KẾ HOẠCH

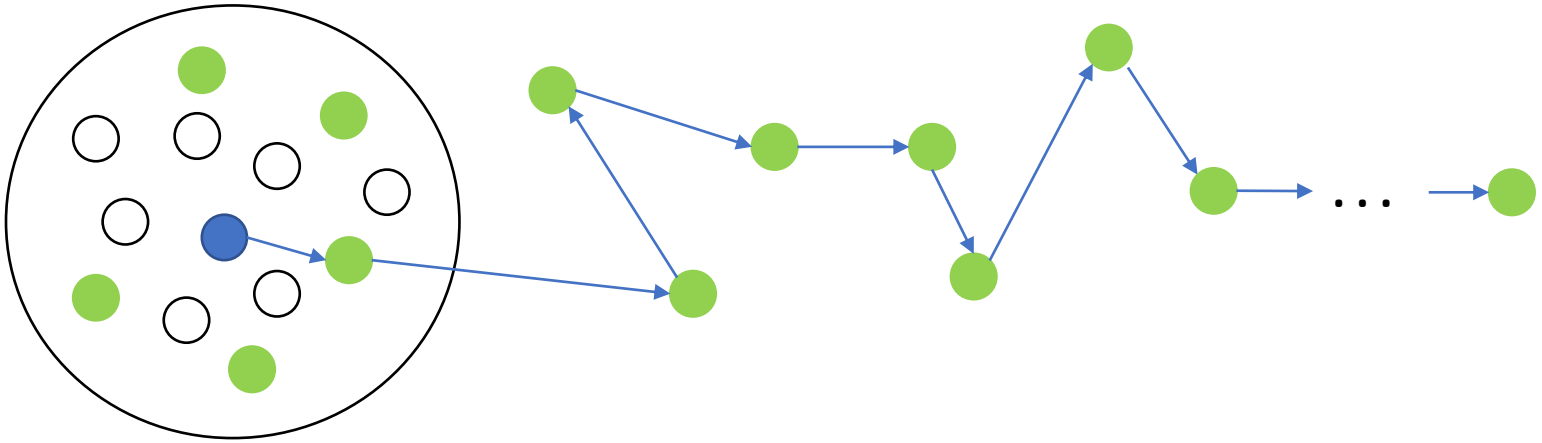
Tìm kiếm cục bộ dựa trên ràng buộc

Nội dung

- Tổng quan phương pháp tìm kiếm cục bộ dựa trên ràng buộc
- Ràng buộc và truy vấn láng giềng
- Heuristic tìm kiếm
- Bài toán N-queen
- Bài toán Sudoku
- Bài toán phân bổ môn học (BACP)
- Bài toán tô màu đồ thị
- Bài toán TSP

Tổng quan

- Tìm kiếm cục bộ
  - Xuất phát từ một lời giải ban đầu (lời giải hiện tại)
  - Mỗi bước lặp
    - Thay thế lời giải hiện tại bằng 1 trong số các lời giải láng giềng
  - Điều kiện kết thúc:
    - Số vòng lặp tối đa
    - Thời gian chạy tối đa cho phép



Ràng buộc và truy vấn láng giềng

- Láng giềng
  - Tập các lời giải được sinh ra từ lời giải hiện tại bằng các áp dụng các toán tử thay đổi cục bộ (local move operators)
  - Phụ thuộc bài toán cụ thể
    - Chọn 1 biến quyết định và gán một giá trị mới
    - Hoán đổi giá trị của 2 biến quyết định
    - Các toán tử thay đổi cục bộ đặc thù cho mỗi lớp bài toán cụ thể: bài toán lập lộ trình vận tải, bài toán lập lịch, . . .

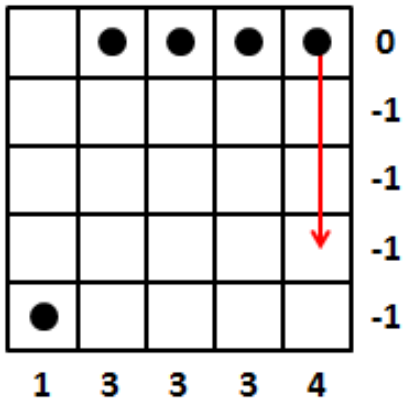
Ràng buộc và lựa chọn láng giềng

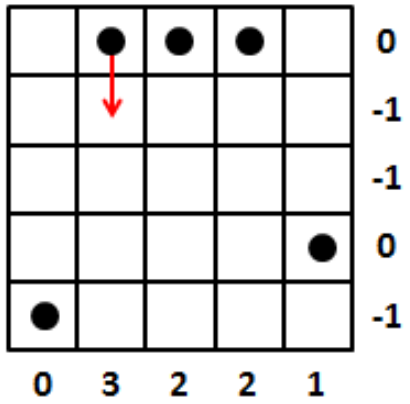
- Ràng buộc và lựa chọn láng giềng
  - Thuật toán tìm kiếm cục bộ tại mỗi bước lặp sẽ lựa chọn 1 lời giải láng giềng và thay thế lời giải hiện tại
  - Sử dụng ràng buộc để ra quyết định lựa chọn
    - Ràng buộc duy trì các thuộc tính thể hiện cấu trúc của bài toán
    - Trong tìm kiếm cục bộ dựa trên ràng buộc (CBLS), mỗi bước lặp lời giải hiện tại sẽ có 1 mức độ vi phạm ràng buộc
    - Tìm kiếm cục bộ sẽ nhắm tới việc tìm lời giải sao cho mức độ vi phạm ràng buộc giảm nhỏ nhất (mức độ vi phạm ràng buộc bằng 0 có nghĩa lời giải đã thỏa mãn ràng buộc)

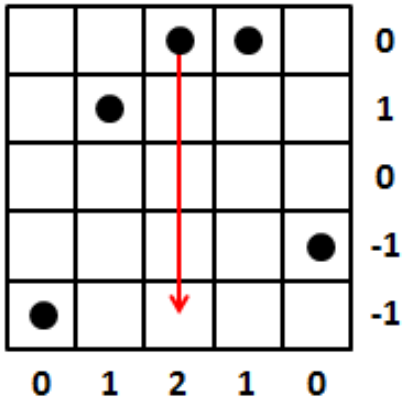
Ràng buộc và lựa chọn láng giềng

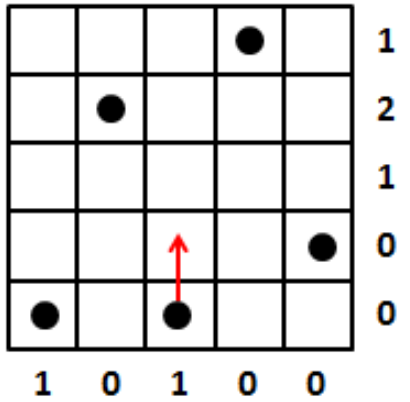
- **Bài toán N-queen**
- Mỗi lời giải
  - Mỗi quân hậu đặt trên 1 cột và trong 1 hàng nào đó
  - Mức độ vi phạm ràng buộc: số cặp 2 quân hậu ăn nhau
- Lời giải ban đầu: đặt ngẫu nhiên n quân hậu trên bàn cờ, mỗi hậu trên 1 cột
- Mỗi bước tìm kiếm cục bộ
  - Chọn 1 quân hậu q ăn nhiều quân hậu khác nhất
  - Chọn 1 hàng mới để di chuyển q đến sao cho tổng mức độ vi phạm ràng buộc giảm nhiều nhất

Ràng buộc và lựa chọn láng giềng

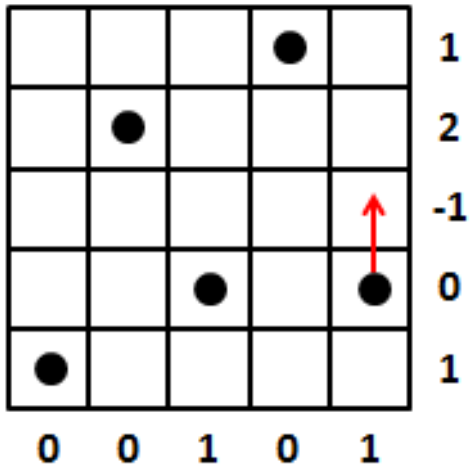


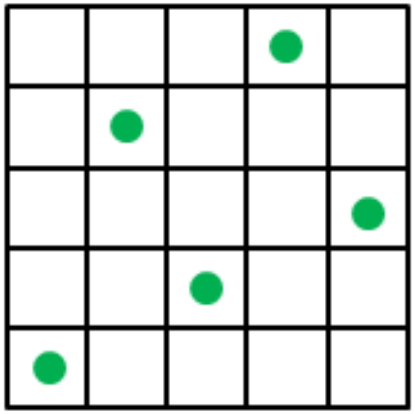






Ràng buộc và lựa chọn láng giềng





Kiến trúc CBLS

- Mô hình
  - Biến quyết định
  - Bất biến
  - Hàm
  - Ràng buộc
- Module tìm kiếm
  - Các chiến lược tìm kiếm heuristics và metaheuristics

Kiến trúc CBLS

- Biến quyết định
  - Biến mô hình hóa lời giải của bài toán
  - Giá trị của các biến xác định lời giải của bài toán

Kiến trúc CBLS

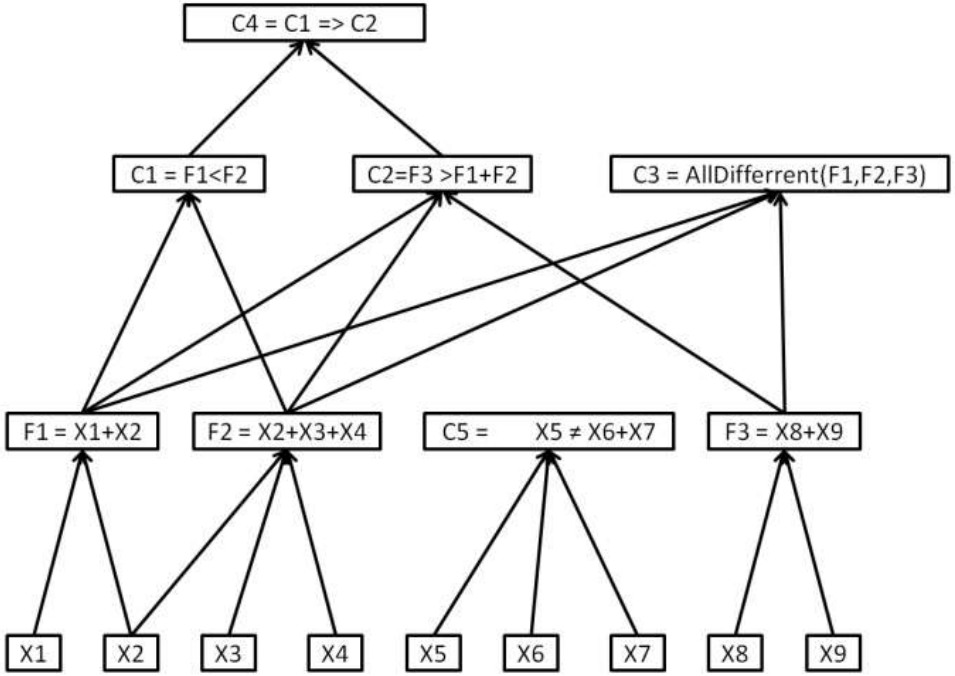
- Bất biến
  - Đối tượng duy trì các thuộc tính của bài toán
  - Phụ thuộc vào biến quyết định, biến quyết định thay đổi giá trị thì các thuộc tính được tự động thay đổi theo
  - Ví dụ
    - $S = X_1 + X_2 + \ldots + X_N$: tổng giá trị các biến
    - $M = \text{argMax}(X_1, X_2, \ldots, X_N)$: tập chỉ số các biến có giá trị lớn nhất

Kiến trúc CBLS

- Hàm, ràng buộc
  - Đối tượng duy trì các thuộc tính của bài toán: mức độ vi phạm ràng buộc
  - Phụ thuộc vào biến quyết định, biến quyết định thay đổi giá trị thì các thuộc tính được tự động thay đổi theo
  - Cho phép truy vấn sự thay đổi của thuộc tính khi biến quyết định thay đổi giá trị (dùng để truy vấn láng giềng)

Kiến trúc CBLS

- Đồ thị phụ thuộc
  - Cấu trúc dữ liệu thể hiện mối quan hệ phụ thuộc giữa các thành phần (biến quyết định, bất biến, hàm, ràng buộc) của mô hình
  - Cho phép cập nhật hiệu quả các thành phần của mô hình mỗi khi biến quyết định thay đổi giá trị (toán tử thay đổi cục bộ)



Kiến trúc CBLS

- Ưu điểm
  - Mô hình và module tìm kiếm độc lập nhau
    - Người dùng có thể thay đổi (thêm vào hoặc loại bỏ ràng buộc khỏi hệ thống ràng buộc) mà không cần thay đổi module tìm kiếm
    - Người dùng có thể áp dụng các chiến lược tìm kiếm khác nhau trên cùng mô hình
  - Thuận tiện nâng cấp mở rộng
    - Người dùng có thể thiết kế và cài đặt các thành phần (hàm, ràng buộc) mới, tuân thủ interface chung và tích hợp vào hệ thống

OpenCBLS - Biến

```
LocalSearchManager mgr = new LocalSearchManager()
```

- Quản lý các thành phần (biến, hàm, ràng buộc của mô hình)
- Lan truyền, cập nhật giá trị các thành phần của mô hình

```
VarIntLS x = new VarIntLS(mgr, min, max)
```

- Biến quyết định, mô hình hóa phương án của bài toán
- Giá trị biến đổi trong quá trình tìm kiếm cục bộ
- Sự thay đổi giá trị của biến sẽ tạo ra sự lan truyền để cập nhật lại giá trị các hàm, ràng buộc định nghĩa trên biến này
- **`min, max`**: giá trị nhỏ nhất và lớn nhất trong miền giá trị của biến

OpenCBLS - Hàm

- Đối tượng được định nghĩa trên các biến quyết định
- Thể hiện các thuộc tính của bài toán

```
interface IFunction{
  int getValue();
  int getAssignDelta(VarIntLS x, int v);
  int getSwapDelta(VarIntLS x, VarIntLS y);
  . . .
}
```

OpenCBLS - Hàm

- Một số hàm cơ bản

Tên hàm	Mô tả
FuncPlus	Cộng
FuncMinus	Trừ
FuncMult	Nhân
Sum	Tổng của 1 mảng các biến, hàm
ConditionalSum	Tổng có điều kiện
Max	Phần tử lớn nhất
Min	Phần tử nhỏ nhất

OpenCBLS - Hàm

- ConditionalSum(VarIntLS[] X, int[] w, int v)
  - Hàm này giúp ích mô hình hóa nhiều bài toán phân công
  - Ý nghĩa: tổng trọng số w[i] tại các chỉ số i sao cho X[i] có giá trị bằng v

```
LocalSearchManager mgr = new LocalSearchManager();
int n = 5;
int[] w = new int[]{3,2,5,4,6};
VarIntLS[] X = new VarIntLS[n];
for(int i = 0; i < n; i++){
X[i] = new VarIntLS(mgr,1,5);
}
X[0].setValue(2);X[1].setValue(1);X[2].setValue(5);
X[3].setValue(2);X[4].setValue(3);
IFunction s = new ConditionalSum(X, w, 3);
mgr.close();
System.out.println("s = " + s.getValue());
```

OpenCBLS - Ràng buộc

- Đối tượng được định nghĩa trên các biến quyết định và các hàm
- Thể hiện các ràng buộc của bài toán

```
interface IConstraint{
  int violations();
  int getAssignDelta(VarIntLS x, int v);
  int getSwapDelta(VarIntLS x, VarIntLS y);
  . . .
}
```

OpenCBLS - Ràng buộc

- Một số ràng buộc cơ bản

Tên ràng buộc	Mô tả
LessOrEqual	Nhỏ hơn hoặc bằng
LessThan	Nhỏ hơn
IsEqual	Bằng
NotEqual	Khác
Implicate	Suy ra
AND	Toán tử AND giữa các ràng buộc
OR	Toán tử OR giữa các ràng buộc
AllDifferent	Ràng buộc khác nhau từng đôi
MultiKnapsack	Ràng buộc về khả năng chứa trong bài toán sắp xếp đối tượng (object) vào các vật chứa (bin)

OpenCBLS - Vi phạm ràng buộc

- Ràng buộc $c \equiv a \leq b$, violations(c) = max{0, a-b}
- Ràng buộc $c \equiv a = b$, violations(c) = $|a-b|$
- Ràng buộc $c \equiv a \neq b$, violations(c) = 0 nếu $a \neq b$ và violations(c) = 1 nếu $a=b$
- Ràng buộc $c \equiv$ **AllDifferent**(x[1..n]), violations(c) = $\Sigma_{v \in D}$max {0, o(v)-1} trong đó o(v) là số lần xuất hiện giá trị v trong x[1..n], với D = Domain(x[1]) $\cup \ldots \cup$ Domain(x[n])

OpenCBLS - Vi phạm ràng buộc

- Ràng buộc $c \equiv$ **MultiKnapsack**(x[1..n], w[1..n], cap[1..m]), violations(c) = $\Sigma_{i\in\{1,\ldots,m\}}\max\{0, L[i]\text{-}cap[i]\}$, trong đó $L[i] = \Sigma_{j\in\{1,\ldots,n\}} (x[j]=i)*w[j]$
- Ràng buộc $c \equiv c_1 \wedge \ldots \wedge c_k$, violations($c$) = violations($c_1$) + … + violations($c_k$)
- Ràng buộc $c \equiv$ ConstraintSystem, violations(c) bằng tổng mức độ vi phạm ràng buộc của các ràng buộc thành phần được đưa vào S
- Ràng buộc $c \equiv c_1 \Rightarrow c_2$, violations($c$) = 0 nếu violations($c_1$) > 0 và violations($c$) = violations($c_2$) nếu violations($c_1$) = 0.

N-queen

- Bài toán n-queen
  - Xếp n quân hậu trên bàn cờ n x n sao cho không có 2 quân hậu nào nằm trên cùng 1 hàng, cùng 1 cột hoặc cùng 1 đường chéo
- Chiến lược tìm kiếm tham lam 2 bước
  - Sinh ra lời giải ban đầu ngẫu nhiên: mỗi quân hậu trên 1 cột
  - Mỗi bước lặp
    - Chọn 1 quân hậu tham gia nhiều vi phạm ràng buộc nhất
    - Di chuyển quân hậu đó đến 1 hàng mới sao cho số vi phạm ràng buộc giảm nhiều nhất

N-queen

```
int N = 10;
LocalSearchManager mgr = new LocalSearchManager();
VarIntLS[] X = new VarIntLS[N];
for(int i = 0; i < N; i++)
X[i] = new VarIntLS(mgr,0,N-1);
ConstraintSystem S = new ConstraintSystem(mgr);

S.post(new AllDifferent(X));

IFunction[] f1 = new IFunction[N];
 for(int i = 0; i < N; i++)
  f1[i] = new FuncPlus(X[i], i);
 S.post(new AllDifferent(f1));
```

N-queen

```
IFunction[] f2 = new IFunction[N];
for(int i = 0; i < N; i++)
  f2[i] = new FuncPlus(X[i], -i);
S.post(new AllDifferent(f2));
mgr.close();
 MinMaxSelector mms = new MinMaxSelector(S);
int it = 0;
while(it < 100000 && S.violations() > 0){
  VarIntLS sel_x = mms.selectMostViolatingVariable();
  int sel_v = mms.selectMostPromissingValue(sel_x);
  sel_x.setValuePropagate(sel_v);// local move

  System.out.println("Step " + it + ", violations = " + S.violations());
  it++;
}
```

Tìm kiếm leo đồi

- Mỗi bước lặp
  - Thu thập tất cả các lời giải láng giềng tốt nhất của lời giải hiện tại vào 1 danh sách *cand*
  - Chọn ngẫu nhiên 1 lời giải *s* từ *cand* và thực hiện thay thế lời giải hiện tại bởi *s*

Tìm kiếm leo đồi

```
class AssignMove{
 int i; // index of variable
 int v; // value to be assigned
 public AssignMove(int i, int v){
  this.i = i; this.v = v;
 }
}
```

```
public class HillClimbingSearch {
 private IConstraint c;
 private ArrayList<AssignMove> cand;
 private VarIntLS[] y;
 private Random R ;

 public HillClimbingSearch(
     IConstraint c){
  this.c = c;
  y = c.getVariables();
  cand = new ArrayList<AssignMove>();
  R = new Random();
 }
 . . .
}
```

Tìm kiếm leo đồi

```
public void exploreNeighborhood(){
 cand.clear();
 int minDelta = Integer.MAX_VALUE;
  for(int i = 0; i < y.length; i++){
   for(int v = y[i].getMinValue();
       v <= y[i].getMaxValue(); v++){
    int d = c.getAssignDelta(y[i], v);
    if(d < minDelta){
       cand.clear();
       cand.add(new AssignMove(i,v));
       minDelta = d;
    }else if(d == minDelta){
       cand.add(new AssignMove(i,v));
    }
   }
  }
 }
```

```
public void search(int maxIter){
  int it = 0;
  while(it < maxIter &&
      c.violations() > 0){
   exploreNeighborhood();
   AssignMove m =
     cand.get(R.nextInt(cand.size()));
   y[m.i].setValuePropagate(m.v);
   it++;
  }
 }
```

Tìm kiếm leo đồi – ví dụ

Bài toán thỏa mãn ràng buộc sau:

- Biến
  - $X = \{X_0, X_1, X_2, X_3, X_4\}$
- Miền giá trị
  - $X_0, X_1, X_2, X_3, X_4 \in \{1,2,3,4,5\}$
- Ràng buộc
  - $C_1: X_2 + 3 \neq X_1$
  - $C_2: X_3 \leq X_4$
  - $C_3: X_2 + X_3 = X_0 + 1$
  - $C_4: X_4 \leq 3$
  - $C_5: X_1 + X_4 = 7$
  - $C_6: X_2 = 1 \Rightarrow X_4 \neq 2$

Tìm kiếm leo đồi – ví dụ

```
LocalSearchManager mgr = new LocalSearchManager();
VarIntLS x = new VarIntLS[5];
for(int i = 0; i < 5; i++)
  x[i] = new VarIntLS(mgr,1,5);
ConstraintSystem S = new ConstraintSystem(mgr);
S.post(new NotEqual(new FuncPlus(x[2],3), x[1]));
S.post(new LessOrEqual(x[3], x[4]));
S.post(new IsEqual(new FuncPlus(x[2],x[3]), new FuncPlus(x[0],1)));
S.post(new LessOrEqual(x[4], 3));
S.post(new IsEqual(new FuncPlus(x[4],x[1]), 7));
S.post(new Implicate(new IsEqual(x[2], 1), new NotEqual(x[4], 2)));
mgr.close();// mandatory

HillClimbingSearch s = new HillClimbingSearch(S);
s.search(10000);
```

Sudoku

- CSP = (X, D, C)
  - Biến: $X = \{X_{1,1}, \ldots, X_{9,9}\}$ trong đó $X_{i,j}$ là giá trị trong ô (i, j), $\forall\ i, j = 1, 2, \ldots, 9$
  - Miền giá trị $D(X_{i,j}) = \{1, \ldots, 9\}$, $\forall\ i, j = 1, 2, \ldots, 9$
  - Ràng buộc
    - Các số trên mỗi cột đôi một khác nhau
      - $X_{i1,j} \neq X_{i2,j}$, với mọi $1 \leq i_1 < i_2 \leq 9$, $1 \leq j \leq 9$
    - Các số trên mỗi hàng đôi một khác nhau
      - $X_{j,i1} \neq X_{j,i2}$, với mọi $1 \leq i_1 < i_2 \leq 9$, $1 \leq j \leq 9$
    - Các số trong mỗi hình vuông con 3x3 đôi một khác nhau
      - $X_{3i+i1,3j+j1} \neq X_{3i+i2,3j+j2}$, với mọi $0 \leq i, j \leq 2$, với mọi $1 \leq i_1, i_2, j_1, j_2 \leq 3$ thỏa mãn $(i_1, j_1) \neq (i_2, j_2)$

Sudoku

- Chiến lược tìm kiếm leo đồi
  - Duy trì tính thỏa mãn ràng buộc trên mỗi hàng
    - Khởi tạo: điền các số từ 1 đến 9 vào 9 ô trên mỗi hàng
  - Local move (bước di chuyển cục bộ)
    - Chọn 2 ô trên cùng 1 hàng và hoán đổi giá trị cho nhau sao cho vi phạm ràng buộc giảm nhiều nhất

Sudoku

```
public class Sudoku {
 LocalSearchManager mgr;// doi tuong quan ly
 VarIntLS[][] X;// bien quyet dinh
 ConstraintSystem S;// he thong cac rang buoc

public void stateModel(){
 mgr = new LocalSearchManager();
 X = new VarIntLS[9][9];
 for(int i = 0; i < 9; i++){
  for(int j = 0; j < 9; j++){
   X[i][j] = new VarIntLS(mgr,1,9);
   X[i][j].setValue(j+1);
  }
 }
 . . .
```

Sudoku

```
S = new ConstraintSystem(mgr);
  // define rang buoc AllDifferent theo hang
  for(int i = 0; i < 9; i++){
   VarIntLS[] y = new VarIntLS[9];
   for(int j = 0; j < 9; j++)
    y[j] = X[i][j];
   S.post(new AllDifferent(y));
  }
  // define rang buoc AllDifferent theo cot
  for(int i = 0; i < 9; i++){
   VarIntLS[] y = new VarIntLS[9];
   for(int j = 0; j < 9; j++)
    y[j] = X[j][i];
   S.post(new AllDifferent(y));
  }
   . . .
```

Sudoku

```
// define rang buoc AllDifferent theo hinh vuong con 3x3
  for(int I = 0; I <= 2; I++){
   for(int J = 0; J <= 2;J++){
    VarIntLS[] y = new VarIntLS[9];
    int idx = -1;
    for(int i = 0; i <= 2; i++)
     for(int j = 0; j <= 2; j++){
      idx++;
      y[idx] = X[3*I+i][3*J+j];
     }
    S.post(new AllDifferent(y));
   }
  }
  mgr.close();
   . . .
```

Sudoku

```
public void search(){
   class Move{
    int i;
    int j1;
    int j2;
    public Move(int i, int j1, int j2){
     this.i = i; this.j1 = j1; this.j2 = j2;
    }
   }
   Random R = new Random();
   ArrayList<Move> candidates = new ArrayList<Move>();
   int it = 0;

   . . .
```

Sudoku

```
while(it <= 100000 && S.violations() > 0){
   candidates.clear(); int minDelta = Integer.MAX_VALUE;
   for(int i = 0; i < 9; i++){
    for(int j1 = 0; j1 < 8; j1++){
     for(int j2 = j1 + 1; j2 < 9; j2++){
      int delta = S.getSwapDelta(X[i][j1], X[i][j2]);
      if(delta < minDelta){
       candidates.clear(); candidates.add(new Move(i,j1,j2));
       minDelta = delta;
      }else if(delta == minDelta)
       candidates.add(new Move(i,j1,j2));
     }
    }
   }
   . . .
```

Sudoku

```
int idx = R.nextInt(candidates.size());
   Move m = candidates.get(idx);
   int i = m.i; int j1 = m.j1; int j2 = m.j2;
   X[i][j1].swapValuePropagate(X[i][j2]);
   it++;
  }
  for(int i = 0; i < 9; i++){
   for(int j = 0; j < 9; j++)
    System.out.print(X[i][j].getValue() + " ");
   System.out.println();
  }
}
```

Sudoku

```
 public static void main(String[] args) {
   Sudoku app = new Sudoku();
   app.stateModel();
   app.search();
  }
}
```

Tìm kiếm tabu

- Ý tưởng chính
  - Duy trì danh sách lưu trữ tbl các bước di chuyển cục bộ gần đây nhất (danh sách Tabu)
  - Tại mỗi bước, không xem xét các bước di chuyển cục bộ trong danh sách Tabu này
  - Aspiration criterion: chấp nhận các bước di chuyển cục bộ nằm trong danh sách Tabu nhưng lại sinh ra lời giải tốt hơn lời giải tốt nhất đã tìm thấy

Tìm kiếm tabu

- Thực thi
  - Ký hiệu **x[]** là biến quyết định
  - Một bước di chuyển cục bộ được đặc trưng bởi 1 cặp **(i,v)** trong đó **x[i]** sẽ được gán lại giá trị **v** để sinh ra lời giải tiếp theo
  - tbl: độ dài của danh sách Tabu
  - Duy trì mảng 2 chiều **tabu[][]** trong đó **tabu[i,v] = k** có nghĩa các bước di chuyển cục bộ **(i,v)** nằm trong danh sách Tabu từ bước lặp hiện tại đến bước lặp k-1
  - Tại mỗi bước lặp **it**:
    - Bước di chuyển cục bộ **(i,v)** được chấp nhận khi **tabu[i,v] <= it** (move này ko nằm trong danh sách Tabu) hoặc việc gán **x[i] = v** sẽ sinh ra lời giải tốt hơn lời giải tốt nhất đã tìm thấy (aspiration criterion)
    - Khi move **(i,v)** được chấp nhận thì thực hiện gán **tabu[i,v] = it + tbl** (để đưa bước di chuyển cục bộ **(i,v)** vào danh sách Tabu

BACP

- Có n môn học 1, 2, …, n, cần phân bổ vào p học kỳ 1, 2, …, p. Môn học i có số tín chỉ là c(i). Giữa các môn học có điều kiện tiên quyết thể hiện bởi cấu trúc Q là tập các bộ (i, j) : môn i phải được xếp vào học kỳ trước học kỳ xếp môn j. Cho trước các hằng số α, β, λ, γ. Hãy lập kế hoạch phân bổ n môn học vào p học kỳ sao cho
- Tổng số môn học trong mỗi học kỳ phải lớn hơn hoặc bằng α và nhỏ hơn hoặc bằng β
- Tổng số tín chỉ các môn học trong mỗi học kỳ phải lớn hơn hoặc bằng λ và nhỏ hơn hoặc bằng γ
- Thỏa mãn ràng buộc về điều kiện tiên quyết

BACP

```
int N = 9;// number of courses: 0,1,2,...,8
int P = 4;// number of semesters: 0,1,2,3
int[] credits = {3, 2, 2, 1, 3, 3, 1, 2, 2};
int alpha = 2;
int beta = 4;
int lamda = 3;
int gamma = 7;
int[] I = {0,0,1,2,3,4,3};
int[] J = {1,2,3,5,6,7,8};// prerequisites


LocalSearchManager mgr = new LocalSearchManager();
VarIntLS[] X = new VarIntLS[N];// X[i] is the semester to which course i is
assigned
for(int i = 0; i < N; i++)
X[i] = new VarIntLS(mgr,0,P-1);
```

BACP

```
ConstraintSystem S = new ConstraintSystem(mgr);
for(int k = 0; k < I.length; k++)
   S.post(new LessThan(X[I[k]],X[J[k]]));
for(int i = 0; i < P; i++){
   ConditionalSum load = new ConditionalSum(X, credits,i);
   S.post(new LessOrEqual(lamda,load));
   S.post(new LessOrEqual(load,gamma));

   ConditionalSum nb = new ConditionalSum(X,i);
   S.post(new LessOrEqual(alpha,nb));
   S.post(new LessOrEqual(nb,beta));
}
mgr.close();

TabuSearch ts = new TabuSearch();
ts.search(S, 20, 10000, 10000, 100);
```

Graph Coloring

```
int N = 6;
int K = 3;
int[][] E = {
  {0,1},
  {0,4},
  {1,2},
  {1,3},
  {1,5},
  {2,5},
  {3,4}
};
LocalSearchManager mgr = new LocalSearchManager();
VarIntLS[] X = new VarIntLS[N];
for(int i = 0; i < N; i++)
X[i] = new VarIntLS(mgr,0,K-1);
```

Graph Coloring

```
ConstraintSystem S = new ConstraintSystem(mgr);


for(int i = 0; i < E.length; i++){
  S.post(new NotEqual(X[E[i][0]],X[E[i][1]]));
}
mgr.close();
TabuSearch ts = new TabuSearch();
ts.search(S, 20, 10000, 10000, 100);


for(int i = 0; i < N; i++)
   System.out.println("X[" + i + "] = " + X[i].getValue());
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

- Các chiến lược kết hợp ràng buộc *c* và hàm mục tiêu *f*

Ví dụ $x = (x[0], ..., x[4])$, trong đó miền giá trị $D(x[i]) = \{1,2,...,5\}$

→ Tìm max $f(x) = 3x[0] + 5x[4]$, thỏa mãn $x[i] \neq x[j]$, với $0 \leq i < j \leq 4$

  - Thuật toán 2 pha
    - Pha thứ nhất: tối ưu ràng buộc
    - Pha thứ hai: duy trì tính thỏa mãn ràng buộc và tối ưu hàm mục tiêu
  - Kết hợp đồng thời ràng buộc và hàm mục tiêu thành 1 hàm duy nhất điều khiển tìm kiếm
    - Hàm 2 thành phần theo thứ tự từ điển: ràng buộc trước, hàm mục tiêu sau: $F = (c, f)$
    - Tổ hợp tuyến tính ràng buộc và hàm mục tiêu
      - $F = \alpha \times c + \beta \times f$

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public class CSPWithObjective {
  class Move{
    int i; int v;
    public Move(int i, int v){
      this.i = i; this.v = v;
    }
  }
  private LocalSearchManager mgr;
  private VarIntLS[] X;
  private IConstraint S;
  private IFunction obj;
  private IFunction F;
  private List<Move> cand = new ArrayList<Move>();
  private Random R = new Random();
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
  public void exploreNeighborhoodConstraint(VarIntLS[] X, IConstraint c, List<Move>
cand) {
    int minDelta = Integer.MAX_VALUE;  cand.clear();
    for(int i = 0; i < X.length; i++){
      for(int v = X[i].getMinValue(); v <= X[i].getMaxValue(); v++){
        int delta = c.getAssignDelta(X[i], v);
        if(delta < 0){
          if(delta < minDelta){
            cand.clear();     cand.add(new Move(i,v));
            minDelta = delta;
          }else if(delta == minDelta){
            cand.add(new Move(i,v));
          }
        }
      }
    }
  }
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void exploreNeighborhoodFunctionMaintainConstraint(VarIntLS[] X,
                    IConstraint c,IFunction f, List<Move> cand) {
  cand.clear(); int minDeltaF = Integer.MAX_VALUE;
  for(int i = 0; i < X.length; i++){
    for(int v = X[i].getMinValue(); v <= X[i].getMaxValue(); v++){
      int deltaC = c.getAssignDelta(X[i], v);
      int deltaF = f.getAssignDelta(X[i], v);
      if(deltaC > 0) continue;// ignore worse constraint violations
      if(deltaF < 0){// consider only better neighbors in hill climbing
        if(deltaF < minDeltaF){
          cand.clear();   cand.add(new Move(i,v));  minDeltaF = deltaF;
        }else if(deltaF == minDeltaF)
          cand.add(new Move(i,v));
      }
    }
  }
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void exploreNeighborhoodFunction(VarIntLS[] X, IFunction f,
                  List<Move> cand) {
  int minDelta = Integer.MAX_VALUE; cand.clear();
  for(int i = 0; i < X.length; i++){
    for(int v = X[i].getMinValue(); v <= X[i].getMaxValue(); v++){
      int delta = f.getAssignDelta(X[i], v);
      if(delta < 0){
        if(delta < minDelta){
          cand.clear();  cand.add(new Move(i,v)); minDelta = delta;
        }else if(delta == minDelta){
          cand.add(new Move(i,v));
        }
      }
    }
  }
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void exploreNeighborhoodConstraintThenFunction(VarIntLS[] X,
                        IConstraint c, IFunction f, List<Move> cand) {
  cand.clear();int minDeltaC = Integer.MAX_VALUE; int minDeltaF = Integer.MAX_VALUE;
  for(int i = 0; i < X.length; i++){
    for(int v = X[i].getMinValue(); v <= X[i].getMaxValue(); v++){
      int deltaC = c.getAssignDelta(X[i], v);int deltaF = f.getAssignDelta(X[i], v);
      if(deltaC < 0 || deltaC == 0 && deltaF < 0){// accept only better neighbors
        if(deltaC < minDeltaC || deltaC == minDeltaC && deltaF < minDeltaF){
          cand.clear(); cand.add(new Move(i,v));
          minDeltaC = deltaC; minDeltaF = deltaF;
        }else if(deltaC == minDeltaC && deltaF == minDeltaF){
          cand.add(new Move(i,v));
        }
      }
    }
  }
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void stateModel(){
  mgr = new LocalSearchManager();
  X = new VarIntLS[5];
  for(int i = 0; i < X.length; i++)
    X[i] = new VarIntLS(mgr,1,5);
  ArrayList<IConstraint> list = new ArrayList<IConstraint>();
  list.add(new AllDifferent(X));

  IConstraint[] arr = new IConstraint[list.size()];
  for(int i = 0;i < list.size(); i++) arr[i]= list.get(i);
  S = new AND(arr);
  obj = new FuncPlus(new FuncMult(X[0], 3), new FuncMult(X[4], 5));
  F = new FuncPlus(new FuncMult(new ConstraintViolations(S), 1000),
           new FuncMult(obj, 1));
  mgr.close();
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void move(){
  Move m = cand.get(R.nextInt(cand.size()));
  X[m.i].setValuePropagate(m.v);
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void search1(){
  int it = 0;
  while(it < 10000 & S.violations() > 0){
    exploreNeighborhoodConstraint(X, S, cand);
    if(cand.size() == 0){
      System.out.println("local optimum"); break;
    }
    move();
    System.out.println("Step " + it + ": violations = " + S.violations() +
          ", obj = " + obj.getValue());
    it++;
  }
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
    System.out.println("Phase 2");
    it = 0;
    while(it < 100){
      exploreNeighborhoodFunctionMaintainConstraint(X, S, obj, cand);
      if(cand.size() == 0){
        System.out.println("local optimum");
        break;
      }
      move();
      System.out.println("Step " + it + ": violations = " + S.violations() +
                  ", obj = " + obj.getValue());
      it++;
     }
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
public void search2(){
  int it = 0;
  while(it < 100){
    exploreNeighborhoodConstraintThenFunction(X, S, obj, cand);
    if(cand.size() ==0){
      System.out.println("local optimum");
      break;
    }
    move();
    System.out.println("Step " + it + ": violations = " + S.violations() +
                ", obj = " + obj.getValue());
    it++;
  }
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

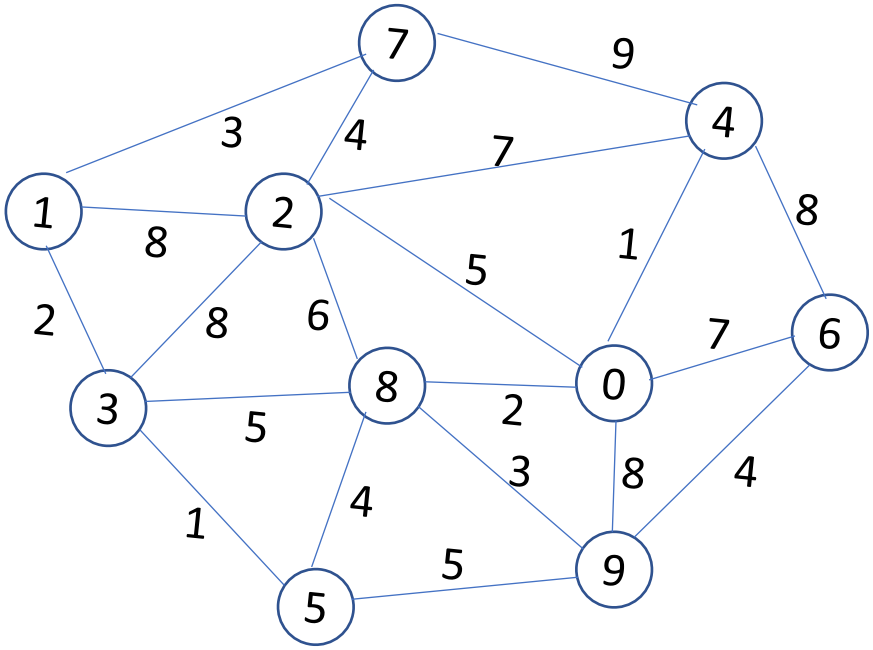
```
public void search3(){
  int it = 0;
  while(it < 100){
    exploreNeighborhoodFunction(X, F, cand);
    if(cand.size() ==0){
       System.out.println("local optimum");
      break;
    }
    move();
    System.out.println("Step " + it + ": violations = " + S.violations() +
              ", obj = " + obj.getValue());
    it++;
  }
}
```

Kết hợp ràng buộc và hàm mục tiêu tối ưu

```
  public static void main(String[] args) {
    // TODO Auto-generated method stub
    CSPWithObjective app = new CSPWithObjective();
    app.stateModel();
    //app.search1();
    //app.search2();
    app.search3();
  }
}
```

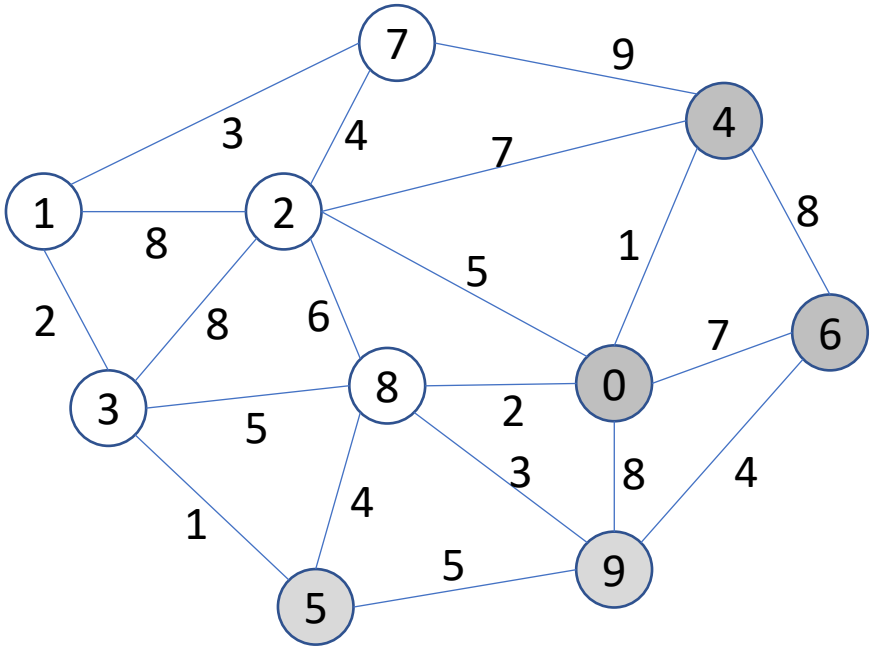
Graph Partitioning

- Cho đồ thị vô hướng *G* = (V,E) trong đó |V| là số chẵn, mỗi cạnh (u,v) của E có trọng số c(u,v)
- Hãy phân hoạch tập đỉnh V thành 2 tập X và Y sao cho
  - |X| = |Y|
  - f = {c(u,v) | u∈X ∧ v∈Y} → min



Graph Partitioning

- Cho đồ thị vô hướng *G* = (V,E) trong đó |V| là số chẵn, mỗi cạnh (u,v) của E có trọng số c(u,v)
- Hãy phân hoạch tập đỉnh V thành 2 tập X và Y sao cho
  - |X| = |Y|
  - f = {c(u,v) | u∈X ∧ v∈Y} → min

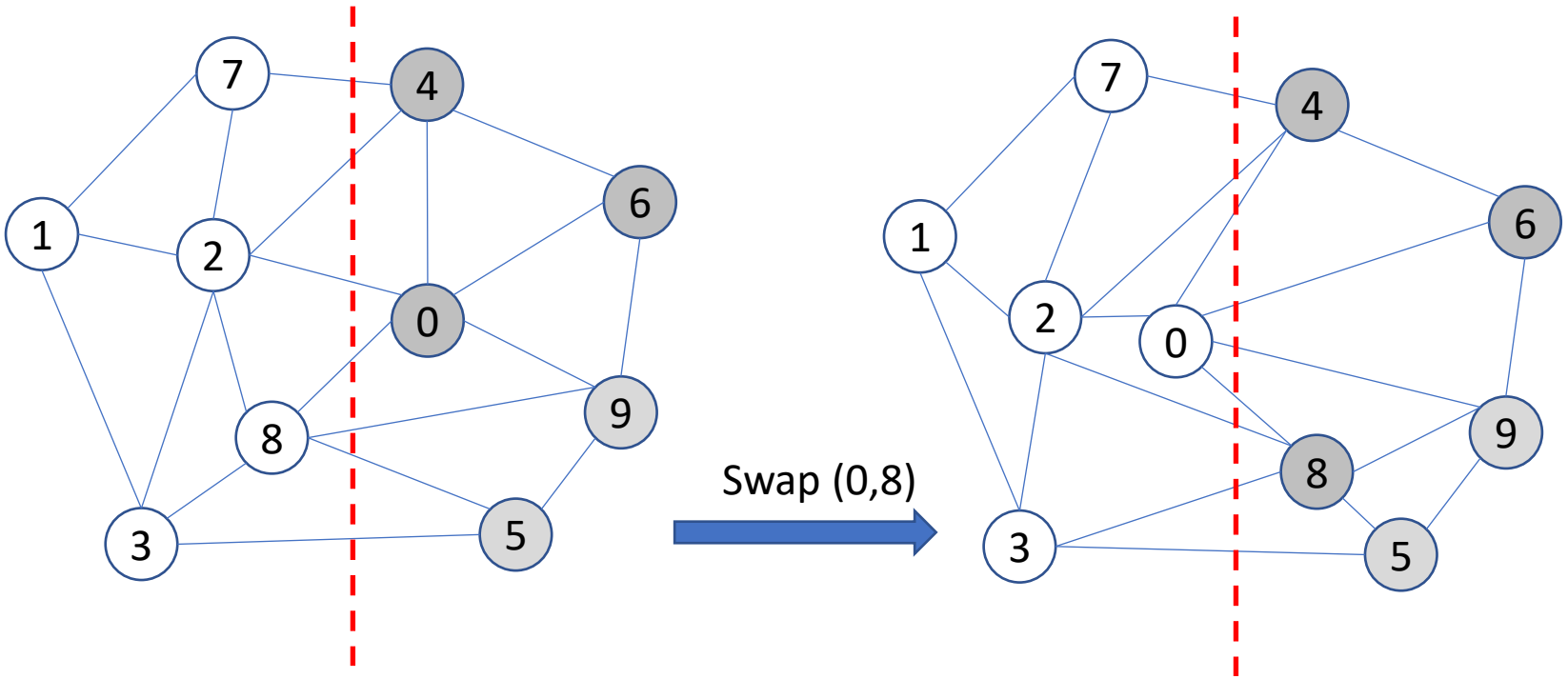


Graph Partitioning

- Mô hình
  - Biến
    - $z(i,j) = 1$: có nghĩa i và j thuộc 2 tập khác nhau
    - $x(i) = 1$: có nghĩa i thuộc tập X, ngược lại $x(i) = 0$ có nghĩa là i thuộc tập Y
  - Ràng buộc
    - $z(i,j) = 1 \rightarrow x(i) + x(j) = 1$
    - $x(i) + x(j) = 1 \rightarrow z(i,j) = 1$
    - $\sum_{i \in V} x(i) = \frac{|V|}{2}$
  - Hàm mục tiêu
    - $f(x,z) = \sum_{(i,j) \in E} c(i,j) z(i,j) \rightarrow$ min

Graph Partitioning

- Duy trì tính thỏa mãn ràng buộc
  - Mỗi bước lặp luôn có 1 phân hoạch với số phần tử bằng nhau
  - Bước di chuyển cục bộ: hoán đổi 2 đỉnh của 2 tập



Graph Partitioning

- Cài đặt hàm cost cho bài toán Graph Partitioning trong OpenCBLS

Graph Partitioning

```
public class GraphPartitioningCost extends AbstractInvariant implements IFunction {
 private LocalSearchManager mgr;
 private VarIntLS[] x;
 private int[][] c;
 private int N; // number of nodes: 0,1,2,...,N-1
 private int value;
 private int minValue;
 private int maxValue;
 private HashMap<VarIntLS, Integer> map;
 private HashSet<Edge>[] A;// A[v] is set of adjacent edges of node v
 class Edge{
  int node;
  int w;
  public Edge(int node, int w){
    this.node = node; this.w = w;
 }
}
```

Graph Partitioning

```
public GraphPartitioningCost(int[][] c, VarIntLS[] x){
  // c cost matrix, (u,v) not an edge -> c[u][v] = 0
  // x: decision variable 0-1
  // check if x is []
  mgr = x[0].getLocalSearchManager(); this.x = x; N = x.length; this.c = c;
  map = new HashMap<VarIntLS, Integer>();
  for(int i = 0; i < N; i++) map.put(x[i],i);
  minValue = 0; maxValue = 0;
  for(int i = 0; i < N; i++)  for(int j = 0; j < N; j++)  maxValue += c[i][j];
  A = new HashSet[N];
  for(int v = 0; v < N; v++){
   A[v] = new HashSet<Edge>();
   for(int u = 0; u < N; u++) if(c[u][v] > 0)
     A[v].add(new Edge(u,c[u][v]));
  }
  mgr.post(this);
}
```

Graph Partitioning

```
@Override
public VarIntLS[] getVariables() {
 return x;
}
@Override
public void propagateInt(VarIntLS z, int val) {
 int old = z.getOldValue();
 if(map.get(z) == null) return;
 if(z.getValue() == old) return;
 int u = map.get(z);// get corresponding node
 for(Edge e: A[u]){
  int v = e.node;
  if(x[u].getValue() == x[v].getValue()) value -= e.w; elsevalue += e.w;
 }
}
```

Graph Partitioning

```
@Override
public void initPropagate() {
 value = 0;
 for(int i = 0; i < N; i++){
   for(int j = i+1; j < N; j++){
     if(x[i].getValue() != x[j].getValue() && c[i][j] > 0)
       value += c[i][j];
   }
 }
}
@Override
public LocalSearchManager getLocalSearchManager() {
  return mgr;
}
```

Graph Partitioning

```
@Override
public int getMinValue() {
 return minValue;
}

@Override
public int getMaxValue() {
 return maxValue;
}
@Override
public int getValue() {
 return value;// current value of the function
}
```

Graph Partitioning

```
@Override
public int getAssignDelta(VarIntLS z, int val) {
  if(map.get(z) == null) return 0;
  int u = map.get(z);// get corresponding node
  if(x[u].getValue() == val) return 0;
  int delta = 0;
  for(Edge e: A[u]){
  int v = e.node;
  if(x[u].getValue() == x[v].getValue()){
   delta += e.w;
  }else{
   delta -= e.w;
  }
 }
 return delta;
}
```

Graph Partitioning

```
@Override
public int getSwapDelta(VarIntLS z, VarIntLS y) {
 if(map.get(z) == null) return getAssignDelta(y,z.getValue());
 if(map.get(y) == null) return getAssignDelta(z,y.getValue());
 int nz = map.get(z); int ny = map.get(y);if(z.getValue() == y.getValue()) return 0;
 int delta = 0;
 for(Edge e: A[nz]){
   int v = e.node; if(v == ny) continue;
   if(x[nz].getValue() == x[v].getValue()) delta += e.w; else delta -= e.w;
 }
 for(Edge e: A[ny]){
   int v = e.node; if(v == nz) continue;
   if(x[ny].getValue() == x[v].getValue()) delta += e.w; else delta -= e.w;
 }
 return delta;
}
}
```

CBLS cho bài toán TSP

```
double EPS = 0.00001;
int N = 5;
int[][] c = {
{0,4,2,5,6},
{2,0,5,2,7},
{1,2,0,6,3},
{7,5,8,0,3},
{1,2,4,3,0}};

ArrayList<Point> clientPoints = new ArrayList<Point>();
ArrayList<Point> allPoints = new ArrayList<Point>();
Point s = new Point(0); Point t = new Point(N);
allPoints.add(s);
for(int i = 1; i < N; i++){
  Point p = new Point(i);
  clientPoints.add(p); allPoints.add(p);
}
allPoints.add(t);
```

CBLS cho bài toán TSP

```
ArcWeightsManager awm = new ArcWeightsManager(allPoints);
for(int i = 0; i < N; i++){
  Point pi = allPoints.get(i);
  for(int j = 0; j < N; j++){
    Point pj = allPoints.get(j);
    awm.setWeight(pi,pj, c[i][j]);
  }
  awm.setWeight(pi, t, c[i][0]);
  awm.setWeight(t, pi, c[0][i]);
}
```

CBLS cho bài toán TSP

```
VRManager mgr = new VRManager();
VarRoutesVR XR = new VarRoutesVR(mgr);
XR.addRoute(s, t);
for(Point p: clientPoints) XR.addClientPoint(p);

TotalCostVR d = new TotalCostVR(XR, awm);
mgr.close();

HashSet<Point> cand = new HashSet<Point>();
for(Point p: clientPoints) cand.add(p);
Point cur = s;
System.out.println("Init XR = " + XR.toString() + ", distance = " +
                          d.getValue());
```

CBLS cho bài toán TSP

```
while(cand.size() > 0){
  double min = Integer.MAX_VALUE;
  Point sel_p = null;
  for(Point p: cand){
    double delta = d.evaluateAddOnePoint(p, cur);
    if(delta < min){
      min = delta;
      sel_p = p;
    }
  }
  mgr.performAddOnePoint(sel_p, cur);
  cur = sel_p;
  cand.remove(sel_p);
  System.out.println("select " + sel_p.ID + ", XR = " + XR.toString() +
                     ", distance = " + d.getValue());
}
```

CBLS cho bài toán TSP

```
while(true){
  double minD = Integer.MAX_VALUE;
  Point sel_x = null; Point sel_y = null;
  for(Point x = XR.startPoint(1); x != XR.endPoint(1); x = XR.next(x)){
    for(Point y = XR.next(x); y != XR.endPoint(1); y = XR.next(y)){
      double delta = d.evaluateTwoOptMoveOneRoute(x, y);
      if(delta < minD){
        minD = delta; sel_x = x; sel_y = y;
      }
    }
  }
  if(minD > -EPS) break;
  mgr.performTwoOptMoveOneRoute(sel_x, sel_y);
  System.out.println("sel_x =  " + sel_x.ID + ", sel_y = " + sel_y.ID +
          ", XR = " + XR.toString() + ", distance = " + d.getValue());
}
```

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

soict.hust.edu.vn/ fb.com/groups/soict