Home Project Report

Spring2006 CS8550

Instructor: Dr. Alex Zelikovsky

Student : Qiong Cheng

MINIMUM CUT LINEAR ARRANGEMENT

Problem

INSTANCE: Graph $G = (V, E)$.

- SOLUTION: A one-to-one function $f : V \to [1..|V|]$.
- MEASURE: Maximum number of cut edges in any integer point, i.e.

  $\max_{i \in [1..|V|]} |\{\{u,v\} \in E : f(u) \le i < f(v)\}|$.
- *Good News:* Approximable within $O(\log |V| \log\log |V|)$ [135].
- *Comment:* Admits a PTAS if $|E| = \Theta(|V|^2)$ [34].
- *Garey and Johnson:* GT44

Problem Formulation

Variables:

$$x_{ij} = \begin{cases} 1 \ \ if \ \ vertex \ \ i \ \ take \ \ position \ \ j \\ \\ 0 \ \ otherwise \end{cases}$$

$$C_{uv} = \begin{cases} 1 \ \ if \ \ there \ \ is \ \ an \ \ edge \ \ between \ \ u \ \ and \ \ v. \\ \\ 0 \ \ otherwise \end{cases}$$

M: the maximum edge cuts for one permutation

Objective Function: Minimize M

Subject to

$\forall j \in [0..|V|-1] \ \ \sum_{i \in V} x_{ij} = 1$

$$\forall i \in V \quad \sum_{j \in [0..|V|-1]} x_{ij} = 1$$

$$\forall (i,j) \in E, u \in [0..n-1], v \in [0..n-1] \;\; if \quad (u <> v) \quad x_{iu} + x_{jv} - 1 \le C_{uv}$$

$$\forall u \in [0..n-1], v \in [0..n-1] \;\; if \quad (u < v) \quad C_{uv} - C_{vu} \ge 0$$

$$\forall k \in [0..n-2] \quad \sum_{\substack{(u,v) \in E \\ u<v \;\; u \le k \;\; k<v}} C_{uv} \le M$$

- **Model Based on Mathprog:**

```
/* Minimum Cut Linear Arrangement  */

/* Written by Qiong Cheng <qcheng1@student.gsu.edu> */

/* The Minimum Cut Linear Arrangement is stated as follows.
   Let a directed graph G = (V, E) be given, where V = {1, ..., n} is
   a set of nodes, E <= V x V is a set of arcs. Let also each arc
   e = (i,j) be assigned a number w[i,j], which is the weight of the
   arc e. The problem is to find a permutation which meets the
   minimum of maximum number of cut edges in any integer point. */

param n, integer;
/* number of nodes */

param m, integer;
/* number of edges */

set V := 0..n-1;
/* set of nodes */

set E, within V cross V;
/* set of arcs */

param W{(i,j) in E};
/* weight from node i to node j */

var M,>=0, <= m;
/* maximum number of cut edges in any integer point (t) */

var X{i in 0..n-1, j in 0..n-1}, binary;
/* X represents as the one-one function; for any i larger than 1 and
less than n,
   a list of reduced choice is a permutation;
   X[i,j] = 1 means that the node i is reduced to j */

var c{u in 0..n-1, v in 0..n-1}, binary;
/* c[i,j] : edge(i,j) belongs to the cut set of k */

minimize MinCut: M;
/* the objective is to find a permutation which meets the minimum
number of
   maximum number of cut edges in any integer point. */
```

```
s.t. oneToConstraint{j in 0..n-1}: (sum{i in 0..n-1} X[i,j]) == 1;
/* constraint: one-to-one function 1 */

s.t. toOneConstraint{i in 0..n-1}: (sum{j in 0..n-1} X[i,j]) == 1;
/* constraint: one-to-one function 2 */

s.t. reductionConstraint{(i,j) in E, u in 0..n-1, v in 0..n-1:u<>v}:
(X[i,u]+X[j,v]-1) <= c[u,v];

s.t. symmetricConstraint{u in 0..n-1, v in 0..n-1:u<v}: c[u,v] -
c[v,u]>=0;

s.t. maxConstraint{k in 0..n-2}: (sum{u in 0..n-1, v in 0..n-1:(u<v and
u<=k and k<v)} c[u,v]) <= M;

solve;

printf "Result : Min Cut Linear Arrangement = %d\n", M;
printf("one-to-one function :\n");
printf{i in 0..n-1, j in 0..n-1} "%d - %d - %d\n", i, j, X[i,j];
printf("------edges in c[i,j]------\n");
printf{i in 0..n-1, j in 0..n-1} "%d - %d - %d\n", i, j, c[i,j];

end;
```

Results:

File : MinCut.dat10

```
param n := 10;
param m := 27;
param : E : W:=
0 9
0 5
0 2
0 1
1 9
1 8
1 7
1 6
1 5
1 4
1 2
2 8
2 7
2 6
2 3
3 9
3 8
3 7
3 6
4 9
4 6
4 5
5 9
5 7
6 8
```

```
6 7
7 8
;
end;
```

- **Results:**

-bash-2.05b$ ./glpsol -m MinCut3.mod -d MinCut.dat10

64 lines were read
Reading data section from MinCut.dat10...
42 lines were read
Generating MinCut...
Generating oneToConstraint...
Generating toOneConstraint...
Generating reductionConstraint...
Generating symmetricConstraint...
Generating maxConstraint...
Model has been successfully generated
lpx_simplex: original LP has 1695 rows, 191 columns, 5325 non-zeros
lpx_simplex: presolved LP has 1694 rows, 191 columns, 5324 non-zeros
lpx_adv_basis: size of triangular part = 1693
      0:   objval =   0.000000000e+00   infeas =   1.000000000e+00 (1)
    132:   objval =   6.927106217e-16   infeas =   4.902170381e-16 (1)
*   132:   objval =   6.927106217e-16   infeas =   1.274564299e-14 (1)
*   133:   objval = -2.674985763e-16   infeas =   1.217744980e-14 (1)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
+   133: mip =   not found yet >=   -inf   (1; 0)
+   519: mip =   1.000000000e+01 >=   0.000000000e+00 100.0% (24; 0)
+ 1288: mip =   9.000000000e+00 >=   0.000000000e+00 100.0% (52; 4)

…..

```
+15782838: mip =   1.000000000e+01 >=   7.333333333e+00  26.7% (455;
1383157)
+15783785: mip =   1.000000000e+01 >=   7.500000000e+00  25.0% (173;
1386666)
+15784433: mip =   1.000000000e+01 >=     tree is empty   0.0% (0;
1389529)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   117485.0 secs
Memory used: 303.9M (318664584 bytes)
Result : Min Cut Linear Arrangement = 10
one-to-one function :
0 -> 2
1 -> 4
2 -> 6
3 -> 7
4 -> 3
5 -> 1
6 -> 8
7 -> 5
8 -> 9
```

```
9 -> 0

Edges:
0 - 1
0 - 2
0 - 3
0 - 4
0 - 5
0 - 7
1 - 0
1 - 2
1 - 3
1 - 4
1 - 5
2 - 0
2 - 1
2 - 4
2 - 6
3 - 0
3 - 1
3 - 4
3 - 5
3 - 8
4 - 0
4 - 1
4 - 3
4 - 5
4 - 6
4 - 8
4 - 9
5 - 0
5 - 1
5 - 3
5 - 6
5 - 7
5 - 8
5 - 9
6 - 2
6 - 4
6 - 5
6 - 7
6 - 8
6 - 9
7 - 0
7 - 5
7 - 8
7 - 9
8 - 5
8 - 9
9 - 5
Model has been successfully processed
```

Coding:

- **Greedy Heuristic Algorithms:**

Randomly select a permutation and calculate the max edge cuts in the permutation; In all heuristic permutations, we get the minimum number as the Min Cut Linear Arrangement result.

```
#ifndef _GHMINCUT_H_
#define _GHMINCUT_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

# define MAX_STR_LENGTH 250
# define isDebug 0
# define isRun 1
#endif //_GHMINCUT_H_

/* Written by Qiong Cheng <qcheng1@student.gsu.edu> */

/* Greedy Heuristic Approach */

/* The Minimum Cut Linear Arrangement is stated as follows.
   Let a directed graph G = (V, E) be given, where V = {1, ..., n} is
   a set of nodes, E <= V x V is a set of arcs. Let also each arc
   e = (i,j) be assigned a number w[i,j], which is the weight of the
   arc e. The problem is to find a permutation which meets the
   minimum of maximum number of cut edges in any integer point. */

void getInputParameter(char *, char *);
void InitializeGraph(char*, int *, int *);
void InitializeInputArray(char *, int **, int);
void InitializeTable(int, int *);
void CreateRandomPermutation(int, int *);
void replaceTable(int, int *, int *);
void GetMaxCutForOnePermutation(int, int *, int *, int *);

int num_of_vertices, num_of_edges, **input_arr;

int main(int argc, char *argv[]){
      int i = 0;
      int iMinCut = 0, iMinCut_T = 0;
      int kMinCut = 0, kMinCut_T = 0;
      int *Fun, *Fun_T;
      long hNum;
      int eEnd1=0, eEnd2=0;
      int eNewE1=0, eNewE2=0;

      if ( argc < 1 ){
            printf("MinCutLA filename\n");
            return 0;
      }

      printf("argc = %d\n", argc);
      for(i = 0; i < argc; i++)
            printf("arg %d: %s\n", i, argv[i]);
```

```
      /* Initialization Phase...*/
      InitializeGraph(argv[1], &num_of_vertices, &num_of_edges);

      input_arr = (int **)malloc(sizeof(int)*num_of_edges);
      for (i=0;i<num_of_edges;i++){
            input_arr[i] = (int *)malloc(sizeof(int)*3);
      }

      InitializeInputArray(argv[1], input_arr,num_of_vertices);

      /* Check the input file...*/
      if (isDebug == 1) printf("num_of_vertices=%d num_of_edges=%d\n",
num_of_vertices, num_of_edges);
      for (i=0;i<num_of_edges;i++){
            if (isDebug == 1) printf("%d %d %d \n",input_arr[i][0],
input_arr[i][1],input_arr[i][2]);
      }

      /*greedy heuristic approach...*/
      printf("Enter the total number of cycles you want to do
heuristics : \n");
    scanf("%d", &hNum);

      /* Initialization */
    Fun_T = (int *)malloc(sizeof(int)*(num_of_vertices));
    Fun = (int *)malloc(sizeof(int)*(num_of_vertices));
      InitializeTable(num_of_vertices, Fun);
      iMinCut = num_of_edges+1;
      kMinCut = 0;
      srand (time(0));

      for (i=0;i<hNum;i++){
            /* Randomly selection as a permutation */
            InitializeTable(num_of_vertices, Fun_T);
            iMinCut_T = 0;
            kMinCut_T = 0;
            CreateRandomPermutation(num_of_vertices, Fun_T);

            /* Get max cut edges for the permutation */
            GetMaxCutForOnePermutation(num_of_vertices, Fun_T,
&iMinCut_T, &kMinCut_T);

            if (iMinCut>iMinCut_T){
                  replaceTable(num_of_vertices, Fun_T, Fun);
                  iMinCut = iMinCut_T;
                  kMinCut = kMinCut_T;
                  if (isDebug == 1) printf(" %d -> %d \n", i, Fun[i] );

            }
      }

      /* Report the result */
      if (isRun == 1) printf(" ===============Report the
result=======\n");
      if (isRun == 1) printf("Optimal Permutation :\n" );
      for (i=0;i<num_of_vertices;i++){
```

```
            if (isRun == 1) printf(" %d -> %d \n", i, Fun[i] );
      }
      if (isRun == 1) printf("----Draw Optimal Graph k=%d MinCut=%d----
--------\n", kMinCut, iMinCut );

      for (i=0;i<num_of_edges;i++){
                  eEnd1 = input_arr[i][0];
                  eEnd2 = input_arr[i][1];

                  eNewE1 = Fun[eEnd1];
                  eNewE2 = Fun[eEnd2];
                  if (isRun == 1) printf("Edge( %d , %d ) \n", eNewE1,
eNewE2 );
      }

    return 0;
}

void GetMaxCutForOnePermutation(int iFTempSize, int *FTemp, int
*iMaxCut, int *kMaxCut){
      int iCutSum=0,k=0,e=0;
      int eEnd1=0, eEnd2=0;
      int eNewE1=0, eNewE2=0;

      if (FTemp == NULL) return;

      for (k=0;k<num_of_vertices-1;k++){
            iCutSum=0;
            for (e=0;e<num_of_edges;e++){
                  eEnd1 = input_arr[e][0];
                  eEnd2 = input_arr[e][1];

                  eNewE1 = FTemp[eEnd1];
                  eNewE2 = FTemp[eEnd2];
                  if (isDebug == 1) printf("%d Edge(%d,%d) -->
Placement(%d,%d)\n", e, eEnd1, eEnd2,eNewE1, eNewE2 );

                  if ( (eNewE1<=k && k<eNewE2) || (eNewE2<=k &&
k<eNewE1) ){
                        iCutSum++;
                  }
            }
            if (isDebug == 1) printf("the Max is %d the k is %d, Max
cut is %d \n", iMaxCut, k, iCutSum);

            if ((*iMaxCut) < iCutSum){
                  *iMaxCut = iCutSum;
                  *kMaxCut = k;
            }
      }

}

void CreateRandomPermutation(int iSize, int *FTable){
      int iLoop = 0, iHigh = iSize-1, iLow=0;
      int rNum;
      int * temp;
```

```
      if (FTable == NULL) return;

    temp = (int *)malloc(sizeof(int)*(iSize));
    InitializeTable(iSize, temp);
      while (iLoop < iSize){
            rNum = iLow + (double)rand () * (iHigh - iLow + 1) /
RAND_MAX;
            if (isDebug == 1) printf("%d\n", rNum);
            if (temp[rNum] == -1){
                  FTable[iLoop++] = rNum;
                  temp[rNum] = iLoop-1;
                  if (isDebug == 1) printf("Function from %d to %d.\n",
iLoop-1, rNum );
            }
      }
      free(temp);
      temp=NULL;
}

void replaceTable(int iSize, int *FTemp, int *FTable){
      int i=0;
      if ( FTable == NULL || FTemp == NULL) return;

      for (i=0;i<iSize;i++){
            FTable[i]=FTemp[i];
      }
}

void InitializeTable(int iSize, int *FTable){
      int i=0;
      if ( FTable == NULL ) return;

      for (i=0;i<iSize;i++){
            FTable[i]=-1;
      }
}

void InitializeGraph(char *inputF, int *num_of_vertices, int
*num_of_edges){
      FILE *f_input_file;
      char line[MAX_STR_LENGTH],ch[16];

      f_input_file = fopen(inputF, "r");
      if (f_input_file == NULL){
            printf("Cannot open input file!");
            exit(1);
      }

      /* Getting the number of vertices, and number of edges parameters
for the input graph */
      fgets(line,MAX_STR_LENGTH,f_input_file);
      getInputParameter(line, ch );
      *num_of_vertices = atoi(ch);

      fgets(line,MAX_STR_LENGTH,f_input_file);
      getInputParameter(line, ch );
```

```
	*num_of_edges = atoi(ch);

	fclose(f_input_file);
}

void getInputParameter(char * line, char * ch ){
	int isFirst = 1,i=0,j=0;
	if (line == NULL || ch == NULL)
		return;

	while (line[i]!='\0'){
		if ( isFirst && line[i] != '=' )
			i++;
		else {
			if ( isFirst == 1 ){
				isFirst = 0;
				i++;
			}

			ch[j++] = line[i++];
		}

	}
	ch[j]='\0';
}

void InitializeInputArray(char * inputF, int **input_arr, int
num_of_vertices){
	FILE *f_input_file;
	char line[MAX_STR_LENGTH],ch[16];
	int graph_parameters[3],i,j,k=0,index=0;

	for (i=0;i<num_of_vertices;i++){
		for (j=0;j<num_of_vertices;j++){
			input_arr[i][j]=0;
		}
	}

	f_input_file = fopen(inputF, "r");
	if (f_input_file == NULL){
		printf("Cannot open input file!");
		exit(1);
	}

	fgets(line,MAX_STR_LENGTH,f_input_file); /* Bypass the first line
with number of vertices, edges...etc */
	fgets(line,MAX_STR_LENGTH,f_input_file); /* Bypass the first line
with number of vertices, edges...etc */

	while (fgets(line,MAX_STR_LENGTH,f_input_file)){
		k = 0;
		i = 0;
		j = 0;
		while (line[i]!='\0'){
			if (line[i]==' '){
				ch[j]='\0';
				graph_parameters[k++] = atoi(ch);
```

```
                        j=0;
                        i++;
                        continue;
                  }

                  ch[j++] = line[i++];
            }

            ch[j]='\0';
            graph_parameters[k] = atoi(ch);

            input_arr[index][0] =graph_parameters[0];
            input_arr[index][1] =graph_parameters[1];
            if (k>=2)
                  input_arr[index][2] =graph_parameters[2];
            else
                  input_arr[index][2] = 1;
            index++;
      }

      fclose(f_input_file);
}
```

- **Results:**

The greedy heuristic algorithm can come up with results rapidly.

For the graph with 10 nodes:(Input the heuristic cycles-10000)

bash-2.05b$ ./GHMinCut g10.txt
argc = 2
arg 0: ./GHMinCut
arg 1: g10.txt
Enter the total number of cycles you want to do heuristics :
10000
iMinCut 28 kMinCut0
iMinCut_T 16 kMinCut_T3
iMinCut 16 kMinCut3
iMinCut_T 14 kMinCut_T5
iMinCut 14 kMinCut5
iMinCut_T 13 kMinCut_T3
iMinCut 13 kMinCut3
iMinCut_T 12 kMinCut_T4
iMinCut 12 kMinCut4
iMinCut_T 11 kMinCut_T1
iMinCut 11 kMinCut1
iMinCut_T 10 kMinCut_T2
==========Report the result=======
Optimal Permutation :
0 -> 7
1 -> 6
2 -> 2

3 -> 0
4 -> 5
5 -> 8
6 -> 3
7 -> 4
8 -> 1
9 -> 9
----**Draw Optimal Graph k=2 MinCut=10-**----------

For the graph with 10 nodes:(Input the heuristic cycles-100000)
bash-2.05b$ ./GHMinCut g20.txt
argc = 2
arg 0: ./GHMinCut
arg 1: g20.txt
Enter the total number of cycles you want to do heuristics :
100000
iMinCut 89 kMinCut0
iMinCut_T 50 kMinCut_T8
--
iMinCut 50 kMinCut8
iMinCut_T 45 kMinCut_T7
--
iMinCut 45 kMinCut7
iMinCut_T 44 kMinCut_T10
--
iMinCut 44 kMinCut10
iMinCut_T 43 kMinCut_T10
--
iMinCut 43 kMinCut10
iMinCut_T 42 kMinCut_T11
--
iMinCut 42 kMinCut11
iMinCut_T 41 kMinCut_T9
--
iMinCut 41 kMinCut9
iMinCut_T 38 kMinCut_T9
--
iMinCut 38 kMinCut9
iMinCut_T 37 kMinCut_T7
--
iMinCut 37 kMinCut7
iMinCut_T 36 kMinCut_T5
--
==========Report the result=======
Optimal Permutation :

0 -> 14
1 -> 6
2 -> 3
3 -> 10
4 -> 17
5 -> 7
6 -> 9
7 -> 1
8 -> 8
9 -> 15
10 -> 16
11 -> 4
12 -> 13
13 -> 0
14 -> 19
15 -> 12
16 -> 18
17 -> 11
18 -> 5
19 -> 2
----**Draw Optimal Graph k=5 MinCut=36**----------

The experiments show the relative error of this heuristic is zero when the number of heuristic cycles is enough large.

However, because the experiments were conducted in the limited amount, the result about relative error is not exact. Additionally, the degree of preciseness of greedy heuristic approach depends on the selection of the number of heuristic cycles.