Investigacion Operativa
Coloreo Particionado de Grafos

21 de noviembre de 2015

Integrante	LU	Correo electrónico
Martin Baigorria	575/14	martinbaigorria@gmail.com
Andrew Ab	???	???

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Resumen: ???

Keywords: ???

Índice

1. Modelo

Dado un grafo $G(V, E)$ con $n = |V|$ vertices y $m = |E|$ aristas, un coloreo de G se define como una asignacion de un color o etiqueta a cada $v \in V$ de forma tal que para todo par de vertices adyacentes $(p, q) \in E$ poseen colores distintos. El clasico problema de *coloreo de grafos* consiste en encontrar un coloreo del grafo que utilize la menor cantidad de colores posibles.

En este trabajo resolveremos una variante de este problema, el *coloreo particionado de grafos*. A partir de un conjunto de vertices V que se encuentra particionado en $V_1, ..., V_k$, el problema consiste en asignar un color $c \in C$ a solo un vertice de cada particion de forma tal que dos vertices adyacentes no reciban el mismo color y minimizando la cantidad de colores utilizados.

Este problema se puede modelar con Programacion Lineal Entera. Para ello, definamos las siguientes variables:

$$x_{pj} = \begin{cases} 1 & \text{si el color } j \text{ es asignado al vertice } p \\ 0 & \text{en caso contrario} \end{cases}$$

$$w_j = \begin{cases} 1 & \text{si } x_{pj} = 1 \text{ para algun vertice } p \\ 0 & \text{en caso contrario} \end{cases}$$

1.1. Funcion objetivo

De esta forma la funcion objetivo del LP consiste en minimizar la cantidad de colores utilizados:

$$min \sum_{j \in C} w_j \tag{1}$$

Notar que $|C|$ esta acotado superiormente por la cantidad de particiones k.

1.2. Restricciones

Los vertices adyacentes no comparten color. Recordar que no necesariamente se le asigna un color a todo vertice.

$$x_{ij} + x_{kj} \leq 1 \quad \forall (i, k) \in E, \ \forall j \in C \tag{2}$$

Solo se le asigna un color a un unico vertice de cada particion $p \in P$. Esto implica que cada vertice tiene a lo sumo solo un color.

$$\sum_{i \in V_p} \sum_{j \in C} x_{ij} = 1 \quad \forall p \in P \tag{3}$$

Si un nodo usa color j, $w_j = 1$:

$$x_{ij} \leq w_j \quad \forall i \in V, \forall j \in C \tag{4}$$

Integralidad y positividad de las variables:

$$x_{ij} \in \{0, 1\} \quad \forall i \in V, \forall j \in C \tag{5}$$

$$w_j \in \{0, 1\} \quad \forall j \in C \tag{6}$$

2. Branch & Bound

3. Desigualdades

4. Cut & Branch

5. Experimentacion

6. Conclusion

7. Apéndice A: Código

7.1. coloring.cpp

```
#include <ilcplex/ilocplex.h>
#include <ilcplex/cplex.h>
#include <string>
#include <vector>

#define TOL 1E-05

ILOSTLBEGIN // macro to define namespace

int main(int argc, char **argv) {

  // Datos de la instancia de dieta
  int n = 3;
  double costo[] = {1.8, 2.3,1.5};
  double calorias[] = {170,50,300};
  double calcio[] = {3,400,40};
  double minCalorias = 2000;
  double maxCalorias = 2300;
  double minCalcio = 1200;
  double maxPan = 3;
  double minLeche = 2;

  // Genero el problema de cplex.
  int status;
  CPXENVptr env; // Puntero al entorno.
  CPXLPptr lp; // Puntero al LP

  // Creo el entorno.
  env = CPXopenCPLEX(&status);

  if (env == NULL) {
    cerr << "Error creando el entorno" << endl;
    exit(1);
  }

  // Creo el LP.
  lp = CPXcreateprob(env, &status, "instancia dieta");


  if (lp == NULL) {
    cerr << "Error creando el LP" << endl;
    exit(1);
  }


  // Definimos las variables. No es obligatorio pasar los nombres de las variables,
     pero facilita el debug.
  // La info es la siguiente:
  double *ub, *lb, *objfun; // Cota superior, cota inferior, coeficiente de la
     funcion objetivo.
  char *xctype, **colnames; // tipo de la variable (por ahora son siempre continuas),
      string con el nombre de la variable.
  ub = new double[n];      // upper bound
```

```
51    lb = new double[n];       // lower bound
52    objfun = new double[n];
53    xctype = new char[n];
54    colnames = new char*[n];
55
56    for (int i = 0; i < n; i++) {
57      ub[i] = CPX_INFBOUND;
58      lb[i] = 0.0;
59      objfun[i] = costo[i];
60      xctype[i] = 'C'; // 'C' es continua, 'B' binaria, 'I' Entera. Para LP (no enteros
            ), este parametro tiene que pasarse como NULL. No lo vamos a usar por ahora.
61      colnames[i] = new char[10];
62    }
63
64    // Nombre de la variable x_m
65    sprintf(colnames[0],"x_m");
66
67    // Nombre de la variable x_l y cota inferior
68    lb[1] = 2;
69    sprintf(colnames[1],"x_l");
70
71    // Nombre de la variable x_p y cota superior
72    ub[2] = 3;
73    sprintf(colnames[2],"x_p");
74
75    // Agrego las columnas.
76    status = CPXnewcols(env, lp, n, objfun, lb, ub, NULL, colnames);
77
78    if (status) {
79      cerr << "Problema agregando las variables CPXnewcols" << endl;
80      exit(1);
81    }
82
83    // Libero las estructuras.
84    for (int i = 0; i < n; i++) {
85      delete[] colnames[i];
86    }
87
88    delete[] ub;
89    delete[] lb;
90    delete[] objfun;
91    delete[] xctype;
92    delete[] colnames;
93
94
95    // CPLEX por defecto minimiza. Le cambiamos el sentido a la funcion objetivo si se
          quiere maximizar.
96    // CPXchgobjsen(env, lp, CPX_MAX);
97
98    // Generamos de a una las restricciones.
99    // Estos valores indican:
100   // ccnt = numero nuevo de columnas en las restricciones.
101   // rcnt = cuantas restricciones se estan agregando.
102   // nzcnt = # de coeficientes != 0 a ser agregados a la matriz. Solo se pasan los
          valores que no son cero.
103
104   int ccnt = 0, rcnt = 3, nzcnt = 0;
105
```

```
106     char sense[] = {'G','L','G'}; // Sentido de la desigualdad. 'G' es mayor o igual y
            'E' para igualdad.
107
108     double *rhs = new double[rcnt]; // Termino independiente de las restricciones.
109     int *matbeg = new int[rcnt]; //Posicion en la que comienza cada restriccion en
            matind y matval.
110     int *matind = new int[3*n]; // Array con los indices de las variables con
            coeficientes != 0 en la desigualdad.
111     double *matval = new double[3*n]; // Array que en la posicion i tiene coeficiente (
             != 0) de la variable cutind[i] en la restriccion.
112
113     // Podria ser que algun coeficiente sea cero. Pero a los sumo vamos a tener 3*n
            coeficientes. CPLEX va a leer hasta la cantidad
114     // nzcnt que le pasemos.
115
116
117     //Restriccion de minimas calorias
118     matbeg[0] = nzcnt;
119     rhs[0] = minCalorias;
120     for (int i = 0; i < n; i++) {
121        matind[nzcnt] = i;
122        matval[nzcnt] = calorias[i];
123        nzcnt++;
124     }
125
126     //Restriccion de maximas calorias
127     matbeg[1] = nzcnt;
128     rhs[1] = maxCalorias;
129     for (int i = 0; i < n; i++) {
130        matind[nzcnt] = i;
131        matval[nzcnt] = calorias[i];
132        nzcnt++;
133     }
134
135     //Restriccion de minimo calcio
136     matbeg[2] = nzcnt;
137     rhs[2] = minCalcio;
138     for (int i = 0; i < n; i++) {
139        matind[nzcnt] = i;
140        matval[nzcnt] = calcio[i];
141        nzcnt++;
142     }
143
144     // Esta rutina agrega la restriccion al lp.
145     status = CPXaddrows(env, lp, ccnt, rcnt, nzcnt, rhs, sense, matbeg, matind, matval,
             NULL, NULL);
146
147     if (status) {
148       cerr << "Problema agregando restricciones." << endl;
149       exit(1);
150     }
151
152     delete[] rhs;
153     delete[] matbeg;
154     delete[] matind;
155     delete[] matval;
156
157     // Seteo de algunos parametros.
158     // Para desactivar la salida poner CPX_OFF.
```

```
159     status = CPXsetintparam(env, CPX_PARAM_SCRIND, CPX_ON);
160
161     if (status) {
162       cerr << "Problema seteando SCRIND" << endl;
163       exit(1);
164     }
165
166     // Por ahora no va a ser necesario, pero mas adelante si. Setea el tiempo
167     // limite de ejecucion.
168     status = CPXsetdblparam(env, CPX_PARAM_TILIM, 3600);
169
170     if (status) {
171       cerr << "Problema seteando el tiempo limite" << endl;
172       exit(1);
173     }
174
175     // Escribimos el problema a un archivo .lp.
176     status = CPXwriteprob(env, lp, "dieta2.lp", NULL);
177
178     if (status) {
179       cerr << "Problema escribiendo modelo" << endl;
180       exit(1);
181     }
182
183     // Tomamos el tiempo de resolucion utilizando CPXgettime.
184     double inittime, endtime;
185     status = CPXgettime(env, &inittime);
186
187     // Optimizamos el problema.
188     status = CPXlpopt(env, lp);
189
190     status = CPXgettime(env, &endtime);
191
192     if (status) {
193       cerr << "Problema optimizando CPLEX" << endl;
194       exit(1);
195     }
196
197     // Chequeamos el estado de la solucion.
198     int solstat;
199     char statstring[510];
200     CPXCHARptr p;
201     solstat = CPXgetstat(env, lp);
202     p = CPXgetstatstring(env, solstat, statstring);
203     string statstr(statstring);
204     cout << endl << "Resultado de la optimizacion: " << statstring << endl;
205     if(solstat!=CPX_STAT_OPTIMAL){
206        exit(1);
207     }
208
209     double objval;
210     status = CPXgetobjval(env, lp, &objval);
211
212     if (status) {
213       cerr << "Problema obteniendo valor de mejor solucion." << endl;
214       exit(1);
215     }
216
```

```
217     cout << "Datos de la resolucion: " << "\t" << objval << "\t" << (endtime - inittime
            ) << endl;
218
219     // Tomamos los valores de la solucion y los escribimos a un archivo.
220     std::string outputfile = "dieta.sol";
221     ofstream solfile(outputfile.c_str());
222
223
224     // Tomamos los valores de todas las variables. Estan numeradas de 0 a n-1.
225     double *sol = new double[n];
226     status = CPXgetx(env, lp, sol, 0, n - 1);
227
228     if (status) {
229       cerr << "Problema obteniendo la solucion del LP." << endl;
230       exit(1);
231     }
232
233
234     // Solo escribimos las variables distintas de cero (tolerancia, 1E-05).
235     solfile << "Status de la solucion: " << statstr << endl;
236     for (int i = 0; i < n; i++) {
237       if (sol[i] > TOL) {
238         solfile << "x_" << i << " = " << sol[i] << endl;
239       }
240     }
241
242
243     delete [] sol;
244     solfile.close();
245
246     return 0;
247 }
```