PRÁCTICA IV

Algoritmo Branch and Bound

Por: Elena Merelo y
Antonio Gámiz

1 Problema

El problema es el descrito en prácticas anteriores, dado un conjunto de n ciudades, encontrar el recorrido cerrado con peso mínimo, es decir, el más corto.

```
struct weight {
  vector< vector<int> > weights;

  weight(int n) {
    weights.resize(n);
    srand(time(NULL));
    for( int i=0; i<n; i++ ) {
      for( int j=n-1-i; j<n; j++ ) {
        weights[i].push_back( rand() % 100 );
      }
    }
  }

  int get(int i, int j) {
    if( i > j) return weights[i][j];
    else if( i < j ) return weights[j][i];
    else return 0;
  }

  int size() { return weights.size(); }
};
```

Listing 1: Matriz de pesos

```
1 struct node{
2   int cs;
3   vector<int> path;
4
5   node(){};
6
7   node(int cs_, vector<int> path_){
8     cs=cs_;
```

```
1 int solution_cost(struct weight & w, vector<int> & v){
2   int cost=0;
3   for( int i=0; i<v.size()-1; i++ )
4     cost+=w.get( v[i], v[i+1] );
5   return cost;
6 }
```

```
1  int min_value_row(int index, vector<int> to_avoid, struct weight & w) {
2    int min=INT_MAX;
3    for( int j=0; j<w.size(); j++ ){
4      if( index!=j && !is_in(to_avoid, j)){ //is_in definida en "auxiliar.cpp"
5        if( w.get(index, j) < min )
6          min=w.get(index, j);
7      }
8    }
9    return min;
10 }
```

```
1  int possible_cost(struct weight & w, vector<int> & v){
2
3    int cost=solution_cost(w, v);
4
5    vector<int> rows_to_calculate_min = supplementary(v, w.size());
6    rows_to_calculate_min.push_back(v.back());
7
8    vector<int> rows_to_avoid = v;
9    rows_to_avoid.erase( rows_to_avoid.begin() );
10
11   for( int i=0; i<rows_to_calculate_min.size(); i++ )
12     cost+=min_value_row(rows_to_calculate_min[i], rows_to_avoid, w);
13
```

```
1 void generate_children(vector<vector<int> > & children, vector<int> & v, int n){
2   children.clear();
3   vector<int> rest=supplementary(v, n);
4   children.resize(rest.size());
5   for(int i=0; i<children.size(); i++){
6     children[i]=v;
7     children[i].push_back(rest[i]);
8   }
9 }
```

Listing 6: Generación de los hijos

```
1 long long int calculated_nodes=0;
2 long long int bounds=0;
3 long long int max_queue_size=0;
```

Listing 8: Variables relativas a complejidad

```
vector<int> branch_and_bound(int root, struct weight & w){

  priority_queue<node, vector<node>, comparison> lnv;
  vector<int> solution;
  struct node current_option;
  vector<vector<int> > children;

  current_option.cs=0;
  current_option.path.push_back(root);
  lnv.push( current_option );

  int min_cost = INT_MAX;

  int aux;
  while( !lnv.empty() ){
    current_option=lnv.top();

    if( max_queue_size < lnv.size() ) max_queue_size=lnv.size();

    lnv.pop();
    if( solution_cost(w, current_option.path) < min_cost ){

      generate_children(children, current_option.path, w.size());

      for(int i=0; i<children.size(); i++){
      calculated_nodes++;
      if( children[i].size() != w.size()){
        if(solution_cost(w, children[i]) < min_cost) lnv.push( node(possible_cost(w,
    children[i]), children[i]) );
        else bounds++;

        }
        else{
          if( solution_cost(w, children[i]) < min_cost ){
            solution=children[i];
            aux=solution_cost(w, children[i]);
            min_cost = ( min_cost < aux ) ? min_cost:aux;
          }
        }
      }
    } else bounds++;
  }
  return solution;
}
```

Listing 9: Algoritmo Branch and Bound

FIN