DM841

Discrete Optimization

Part 2 – Lecture 1
Local Search
Overview

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Combinatorial Optimization
2. Vertex Coloring
3. Heuristic Methods
   - Local Search

Outline

1. Combinatorial Optimization

2. Vertex Coloring

3. Heuristic Methods
   - Local Search

General vs Instance

General problem *vs* problem instance:

General problem Π:

- Given *any* set of points X in a square, find a shortest Hamiltonian cycle
- *Solution:* Algorithm that finds shortest Hamiltonian cycle for any X

Problem instantiation $\pi = \Pi(I)$:

- Given a specific set of points I in the square, find a shortest Hamiltonian cycle
- *Solution:* Shortest Hamiltonian cycle for I

Problems can be formalized on sets of problem instances $\mathcal{I}$ (instance classes)

Traveling Salesman Problem

Types of TSP instances:

- Symmetric: For all edges uv of the given graph G, vu is also in G, and $w(uv) = w(vu)$.
  Otherwise: asymmetric.
- Euclidean: Vertices = points in an Euclidean space,
  weight function = Euclidean distance metric.
- Geographic: Vertices = points on a sphere,
  weight function = geographic (great circle) distance.

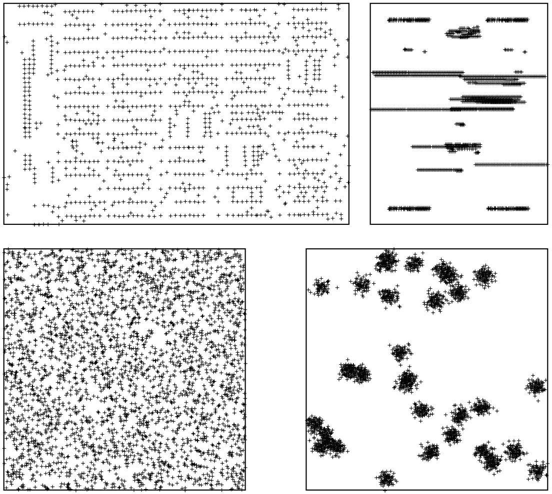
TSP: Benchmark Instances

Instance classes

- Real-life applications (geographic, VLSI)
- Random Euclidean
- Random Clustered Euclidean
- Random Distance

Available at the TSPLIB (more than 100 instances upto 85.900 cities) and at the 8th DIMACS challenge

TSP: Instance Examples



Outline

1. Combinatorial Optimization

2. Vertex Coloring

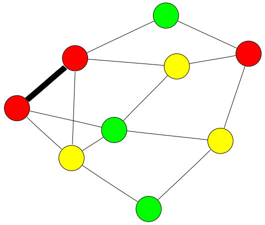
3. Heuristic Methods
   Local Search

The Vertex Coloring Problem

Given: A graph G and a set of colors Γ.

A proper coloring is an assignment of one color to each vertex of the graph such that adjacent vertices receive different colors.

Decision version (k-coloring)
Task: Find a proper coloring of G that uses at most k colors.
Optimization version (chromatic number)
Task: Find a proper coloring of G that uses the minimal number of colors.



Design an algorithm for solving general instances of the graph coloring problem.

Exercise

Map coloring:

Constraint Programming

- Model
  - Parameters
  - Variables and Domains
  - Constraints
  - Objective Function
- Search (solve a decision problem)
  - Branching
    - Variable selection
    - Value selection
  - Search strategy
    - BFS
    - DFS
    - LDS

CP-model

CP formulation:

$$
\begin{array}{rll}
\textit{variables}: & \texttt{domain}(\texttt{y}_\texttt{i}) = \{1, \dots, K\} & \forall i \in V \\
\textit{constraints}: & y_i \neq y_j & \forall ij \in E(G) \\
& \texttt{alldifferent}(\{\texttt{y}_\texttt{i} \mid \texttt{i} \in \texttt{C}\}) & \forall C \in \mathcal{C}
\end{array}
$$

Propagation: An Example

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	(R)	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	(R)	B	(G)	R B	R G B	B	R G B
After *V=blue*	(R)	B	(G)	R	(B)		R G B

Figure 5.6 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Local Search

- Model
  - Variables $\rightsquigarrow$ solution representation, search space
  - Constraints:
    - implicit
    - one-way defining invariants
    - soft
  - evaluation function
- Search (solve an optimization problem)
  - Construction heuristics
  - (Stochastic) local search, metaheuristics
    - Neighborhoods
    - Iterative Improvement
    - Tabu Search
    - Simulated Annealing
    - Guided Local Search
  - Population based metaheuristics

$$\begin{aligned} \textit{variables}: &\quad \texttt{domain}(\texttt{y}_\texttt{i}) = \{1, \ldots, K\} & \forall i \in V \\ \textit{constraints}: &\quad y_i \neq y_j & \forall ij \in E(G) \end{aligned}$$

```
range Vertices = 1..nv;
range Colors = 1..nv;
int nbc = Colors.getUp();

LS m;
Var<int> y[Vertices](m, Colors) := 1;

ConstraintSystem S(m);
forall (i in Vertices, j in Vertices: j>i && adj[i,j])
    S.post(y[i] != y[j]);
```

```
// CONSTRUCTION HEURISTIC
set{int} dom[v in Vertices] = setof(c in Colors) true;
RandomPermutation perm(Vertices);
forall (i in 1..nv) {
  int v = perm.get();
  selectMin(c in dom[v])(c) {
    y[v] := c;
    forall(w in Vertices: adj[v,w])
      dom[w].delete(c);
  }
}
nbc = max(v in Vertices) y[v];
Colors = 1..nbc;
cout<<"Construction heuristic, done: "<<nbc<<" colors"<< endl;
```

```
Solution bestsol = new Solution(m);
int itLimit = 1000*Vertices.getUp();
int maxidle = 10*Vertices.getUp();
int it = 0;
int idle = 0;

int best = S.violations();
while (S.violations() > 0 && idle < maxidle && it < itLimit) {
    selectMin(v in Vertices, c in Colors, d = S.getAssignDelta(col[v],c)) (d)
    {
        // cout<<it<<" v:"<<v<<" c:"<<c<<" "<<S.getAssignDelta(col[v],c)<<endl;
        col[v] := c;
    }
    if ( violations < best)
    {
        // cout<<"+";
        best = violations;
        idle=0;
    }
    else
    {
        // cout<<"-";
        idle++;
    }
    it++;
}
// cout<<it<<" "<<idle<<endl;
cout<<"final: "<<max(v in Vertices) col[v]<<endl;
```

Guidelines for an analysis

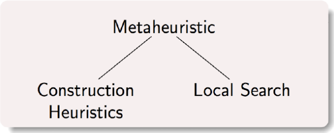
- Given that a feasible coloring exists, is there always a non-null probablity to find it from any initial solution?
- Will the procedure repeat the same moves and/or solutions? Will it end or will it loop?
- Are we doing unecessary work?
- Are we returning a local optimum?

Outline

Heuristics

Get inspired by approach to problem solving in human mind
[A. Newell and H.A. Simon. "Computer science as empirical inquiry: symbols and search." Communications of the ACM, ACM, 1976, 19(3)]

- effective rules
- trial and error



Applications:

- Optimization
- But also in Psychology, Economics, Management [Tversky, A.; Kahneman, D. (1974). "Judgment under uncertainty: Heuristics and biases". Science 185]

Basis on empirical evidence rather than mathematical logic. Getting things done in the given time.

Outline

Local Search

Main idea for combinatorial optimization

- Sequential modification of a small number of decisions
- Incremental evaluation of solutions, generally in O(1) time (Differentiable Objects in Van Hentenryck and Michel's book)
  - Lazy propagation of constraints
  - Usage of invariants

  $\rightsquigarrow$ Small improvement probability but small time and space complexity
  $\rightsquigarrow$ Millions of moves per minute
- (Meta)heuristic rules to drive the search

Local Search Modeling

Can be done within the same framework of Constraint Programming.
See Constraint Based Local-Search (Van Hentenryck and Michel).

- Decide the variables.
  An assignment of these variables should identify a candidate solution or a candidate solution must be retrievable efficiently
  Must be linked to some Abstract Data Type (arrays, sets, permutations).
- Express the implicit constraints on these variables
- Relax some constraints that are difficult to satisfy to become soft constraints
- Express the evaluation function to handle soft constraints and objective function

No restrictions are posed on the language in which the above elements are expressed.