DM811
Heuristics for Combinatorial Optimization

Examples

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Examples

Iterative Improvement for TSP

TSP-2opt-first(s)
input: *an initial candidate tour* $s \in S(\in)$
output: *a local optimum* $s \in S_\pi$

for $i = 1$ to $n - 1$ **do**
  for $j = i + 1$ to n **do**
    if $P[i] + 1 = P[j]$ or $P[j] + 1 = P[i]$ **then** *continue*
    if $P[i] + 1 \geq n$ or $P[j] + 1 \geq n$ **then** *continue*

$$\begin{aligned}\Delta_{ij} \quad = \quad & d(\pi_{P[i]}, \pi_{P[j]}) + d(\pi_{P[i]+1}, \pi_{P[j]+1}) + \\ & -d(\pi_{P[i]}, \pi_{P[i]+1}) - d(\pi_{P[j]}, \pi_{P[j]+1})\end{aligned}$$

    if $\Delta_{ij} < 0$ **then**
      $\mathtt{UpdateTour(s, i, j)}$

is it really?

Examples

Iterative Improvement for TSP

TSP-2opt-first(s)
input: *an initial candidate tour* $s \in S(\in)$
output: *a local optimum* $s \in S_\pi$

Improvement:=$\mathtt{TRUE}$;
while Improvement is `TRUE` **do**
  Improvement:=$\mathtt{FALSE}$;
  for $i = 1$ to $n - 1$ **do**
    for $j = i + 1$ to n **do**
      if $P[i] + 1 = P[j]$ or $P[j] + 1 = P[i]$ **then** *continue*
      if $P[i] + 1 \geq n$ or $P[j] + 1 \geq n$ **then** *continue*

$$\begin{aligned}\Delta_{ij} \quad = \quad & d(\pi_{P[i]}, \pi_{P[j]}) + d(\pi_{P[i]+1}, \pi_{P[j]+1}) + \\ & -d(\pi_{P[i]}, \pi_{P[i]+1}) - d(\pi_{P[j]}, \pi_{P[j]+1})\end{aligned}$$

      if $\Delta_{ij} < 0$ **then**
        $\mathtt{UpdateTour}(\mathtt{s}, \mathtt{i}, \mathtt{j})$
        Improvement=$\mathtt{TRUE}$

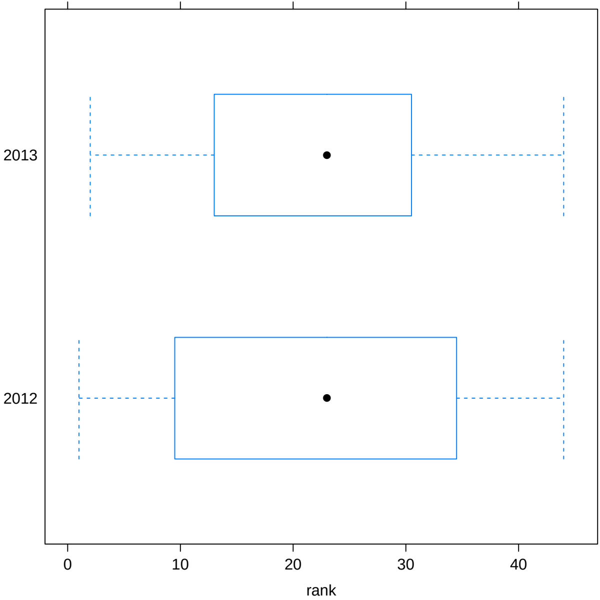
Summary: Local Search Algorithms
(as in [Hoos, Stützle, 2005])

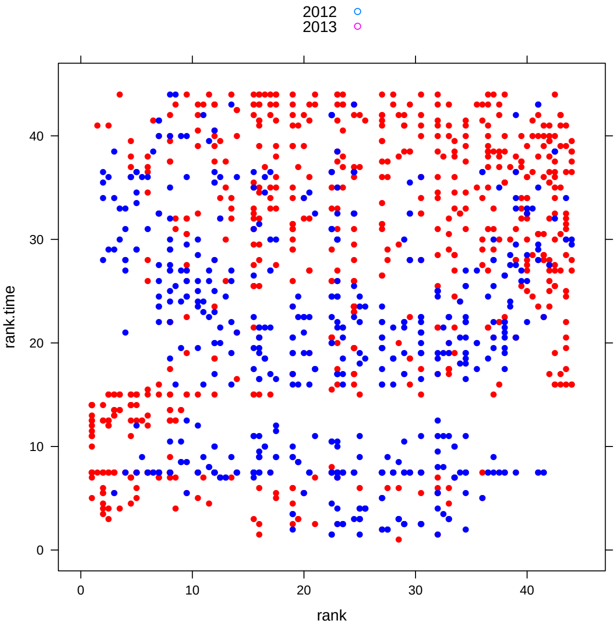
For given problem instance π:

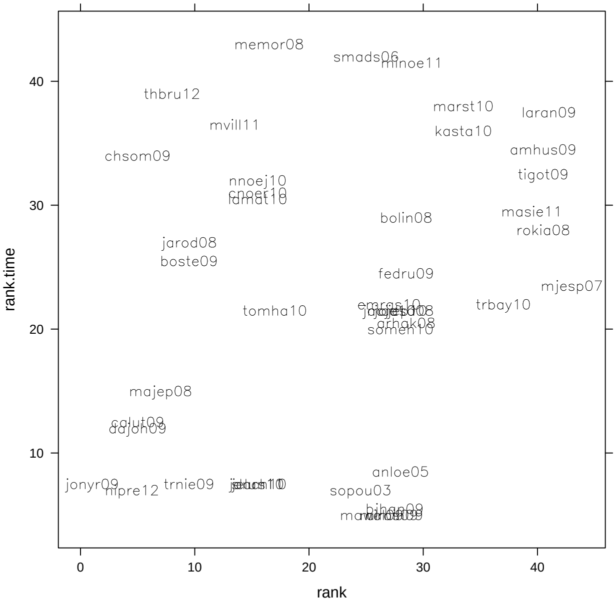
1. search space S_π
2. evaluation function $f_\pi : S \to \mathbf{R}$
3. neighborhood relation $\mathcal{N}_\pi \subseteq S_\pi \times S_\pi$
4. set of memory states M_π
5. initialization function $\texttt{init} : \emptyset \to S_\pi \times M_\pi)$
6. step function $\texttt{step} : S_\pi \times M_\pi \to S_\pi \times M_\pi$
7. termination predicate $\texttt{terminate} : S_\pi \times M_\pi \to \{\top, \bot\}$

Outline

1. GCP
    - CH for GCP
    - Code


2013
2012
0
10
20
30
40
rank


2012
2013
rank.time
rank
0
10
20
30
40


memor08
smads06
minoe11
thbru12
marst10
laran09
mvill11
kasta10
amhus09
chsom09
tigot09
nnoej10
cnoer10
lamat10
masie11
bolin08
rokia08
jarod08
boste09
fedru09
mjesp07
trbay10
tomha10
arhak08
someh10
majep08
calut09
dajon09
anloe05
jonyr09
nipre12
trnie09
sopou03
bihan09
rank.time
rank
0
10
20
30
40

Construction Heuristics

- sequential heuristics
  1. choose a variable (vertex)
     a) static order: random (ROS),
        largest degree first, smallest degree last
     b) dynamic order: saturation degree (DSATUR) [Brélaz, 1979]
  2. choose a value (color): greedy heuristic

```
Procedure ROS
RandomPermutation π(Vertices);
forall i in 1, ..., n do
    v := π(i);
    select min{c : c not in saturated[v]};
    col[v] := c;
    add c in saturated[w] for all w adjacent v;
```

$\mathcal{O}(nk + m) \rightsquigarrow \mathcal{O}(n^2)$

```
Procedure DSATUR
select vertex v uncolored with max degree;
while uncolored vertices do
    select min{c : c not in saturated[v]};
    col[v] := c;
    add c in saturated[w] for all w adjacent v;
    select uncolored v with max size of
      saturated[v];
```

$\mathcal{O}(n(n + k) + m) \rightsquigarrow \mathcal{O}(n^2)$

- partitioning heuristics
  - recursive largest first (RLF) [Leighton, 1979]
    iteratively extract stable sets

Alternative form of pseudo-code

Procedure ROS
RandomPermutation π(Vertices);
forall i in $1, \ldots, n$ **do**
    v := π(i);
    selectMin {c : c not in saturated[v]} **do**
        col[v] := c;
        forall w in Vertices: adj[v,w] **do**
            saturated[w].insert(c);

Procedure DSATUR
RandomPermutation π(Vertices);
forall i in $1, \ldots, n$ **do**
    v := π(i);
    selectMin {c : c not in saturated[v]} **do**
        col[v] := c;
        forall w in Vertices: adj[v,w] **do**
            saturated[w].insert(c);

RLF [Leighton, 1979]

Procedure Recursive Largest First(G)
In $G = (V, E)$: input graph;
Out k : upper bound on $\chi(G)$;
Out c : a coloring $c : V \mapsto K$ of G;

$k \leftarrow 0$ **while** $|V| > 0$ **do**
  $k \leftarrow k + 1$ /* Use an additional color */
  FindStableSet(V, E, k) /* $G = (V, E)$ is reduced */
return k

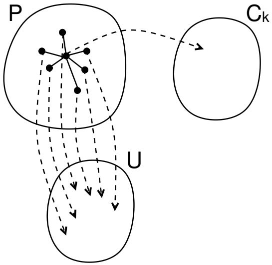
RLF

Key idea: extract stable sets trying to maximize edges removed.

Procedure FindStableSet(G, k)
In $G = (V, E)$: input graph
In k : color for current stable set
Var P : set of potential vertices for stable set
Var U : set of vertices that cannot go in current stable set

$P \leftarrow V$; $U \leftarrow \emptyset$;
forall $v \in P$ **do** $d_U(v) \leftarrow 0$; /* degree induced by U */
while P not empty **do**
    select v in P with max d_U;
    move v from P to C_k; $V \leftarrow V \setminus \{v\}$
    forall $w \in \delta_P(v)$ **do** /* neighbors of v in P */
        move w from P to U; $E \leftarrow E \setminus \{v, w\}$
        forall $u \in \delta_P(w)$ **do**
            $d_U(u) \leftarrow d_U(u) + 1$

$$\mathcal{O}(m + n\Delta^2) \rightsquigarrow \mathcal{O}(n^3)$$



Examples

```
import cotls;
include "loadDIMACS";
// int nv;
// int me;
// float alpha;
// bool adj[nv,nv];
range Vertices = 1..nv;
range Colors = 1..nv;
int nbc = Colors.getUp();

Solver<LS> m();

var{int} col[Vertices](m,Colors) := 1;
ConstraintSystem<LS> S(m);

forall (i in Vertices, j in Vertices: j>i && adj[i,j])
S.post(col[i] != col[j]);
S.close();

m.close();

// 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) {
      col[v] := c;
      forall(w in Vertices: adj[v,w])
         dom[w].delete(c);
   }
}
nbc = max(v in Vertices) col[v];
Colors = 1..nbc;
cout<<"Construction heuristic, done: "<<nbc<<" colors"<< endl;
```

code1.java/png code3.cpp

References

Brélaz D. (1979). **New methods to color the vertices of a graph**. *Communications of the ACM*, 22(4), pp. 251–256.

Leighton F.T. (1979). **A graph coloring algorithm for large scheduling problems**. *Journal of Research of the National Bureau of Standards*, 84(6), pp. 489–506.