Local search with perturbations for the prize collecting Steiner tree problem in graphs

Celso C. Ribeiro

Catholic University of Rio de Janeiro, Brazil

(on leave at Algorithms & Optimization Research Dept., AT&T Labs Research, Florham Park, NJ)

celso@research.att.com

http://www.inf.puc-rio.br/~celso

Joint work with S. Canuto & M.G.C. Resende

April 2001

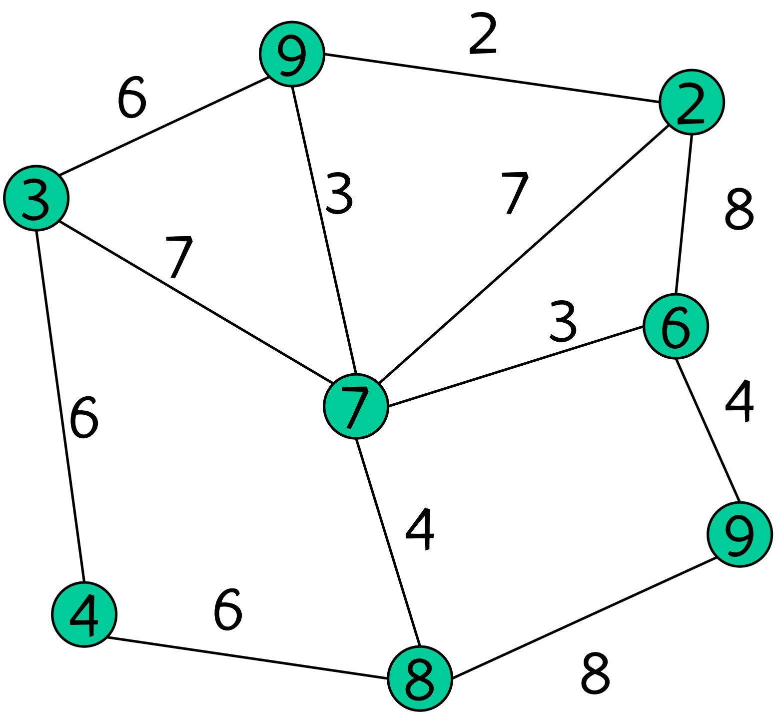
Outline

- Introduction
  - Problem definition
  - An application from telecommunications access network design
- Local search with perturbations
  - Local search with perturbations
  - Path relinking
  - Variable neighborhood search
- Computational results

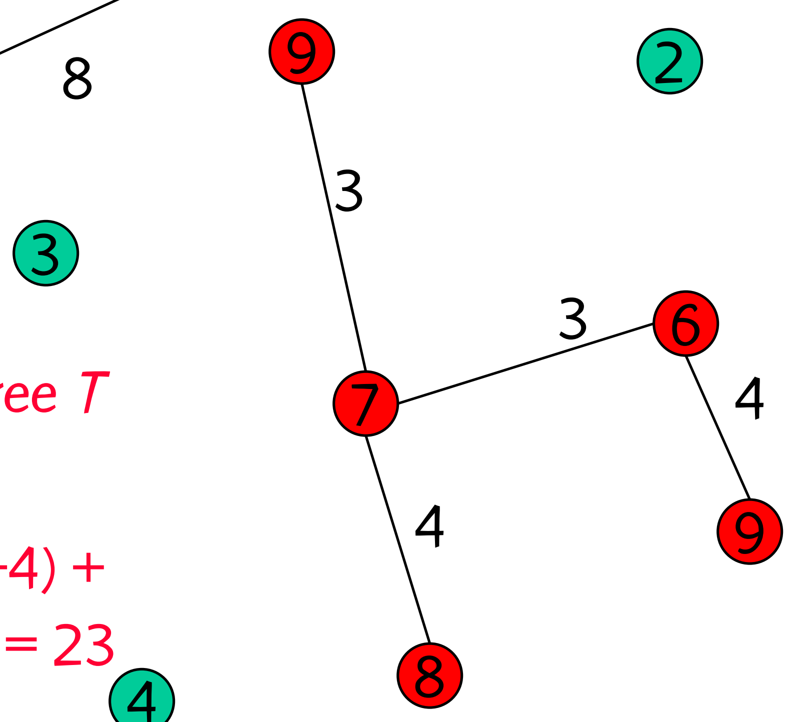
Prize-collecting Steiner tree (PCST) problem

- Given: graph $G = (V, E)$
  - Real-valued cost c_e is associated with edge e
  - Real-valued penalty d_v is associated with vertex v
- *A* tree is a connected acyclic subgraph of G and its weight is the sum of its edge costs plus the sum of the penalties of the vertices of G not spanned by the tree.
- PCST problem: Find tree of smallest weight.

Cost of tree



graph G



tree T

Cost $(T) = (3+3+4+4) +$
$(2+3+4) = 23$

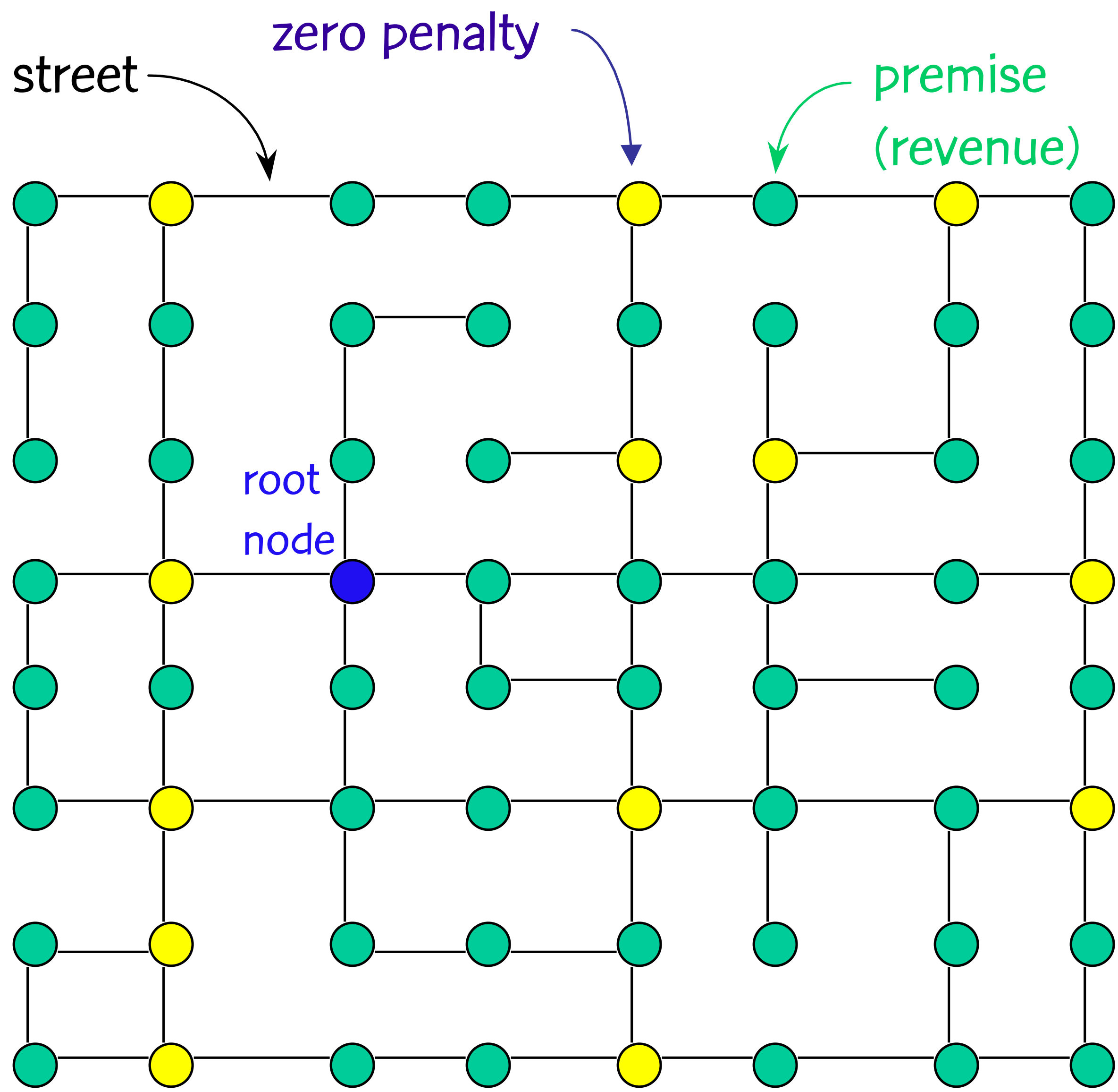
Design of local access telecommunications network

- Build a fiber-optic network for providing broadband connections to business and residential customers.
- Design a local access network taking into account tradeoff between:
  - cost of network
  - revenue potential of network

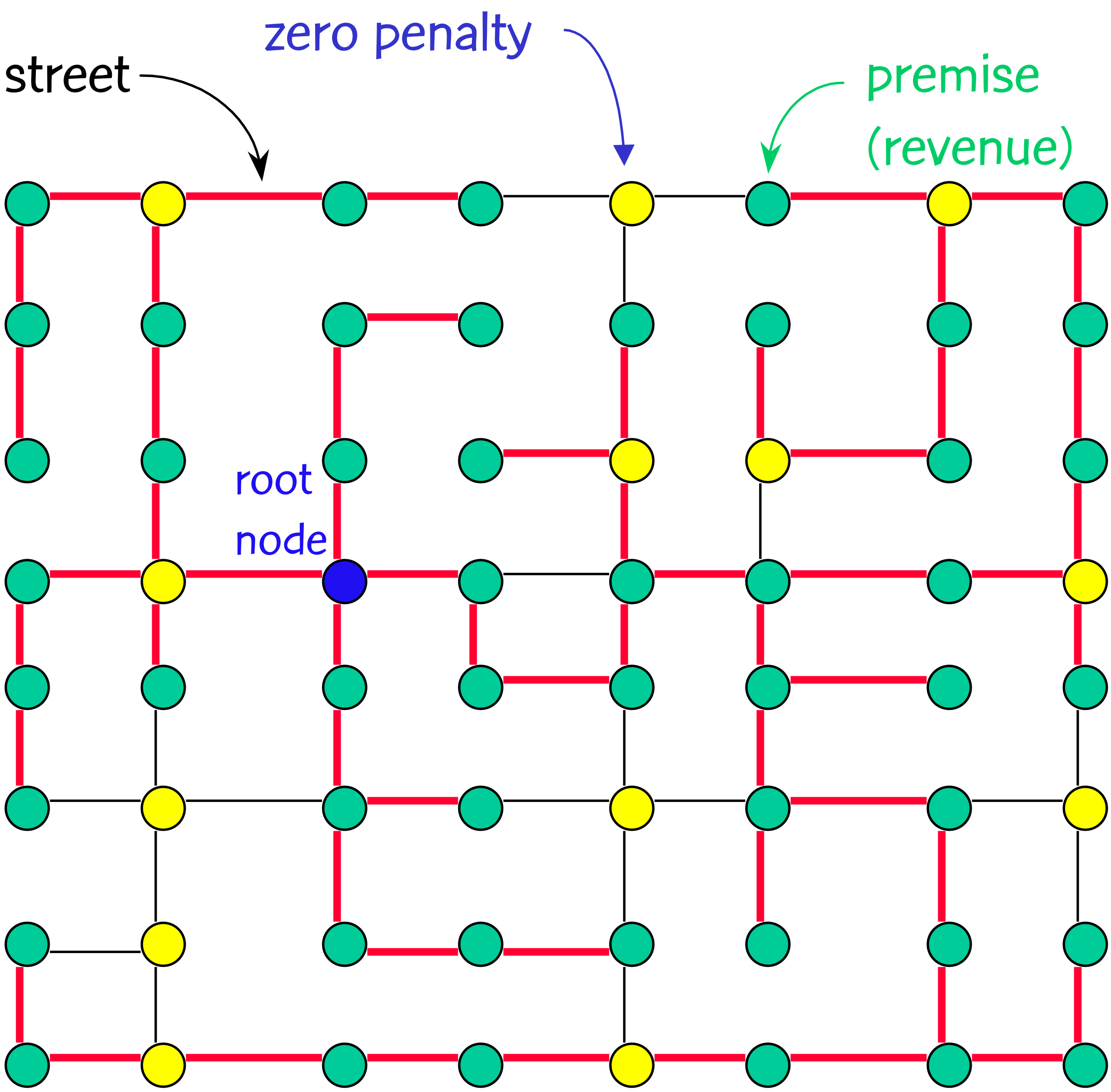
Design of local access telecommunications network

- Graph corresponds to local street map
  - Edges: street segments
    - Edge cost: cost of laying the fiber on the corresponding street segment
  - Vertices: street intersections and potential customer premises
    - Vertex penalty: estimate of potential loss of revenue if the customer were not to be serviced (intersection nodes have no penalty)

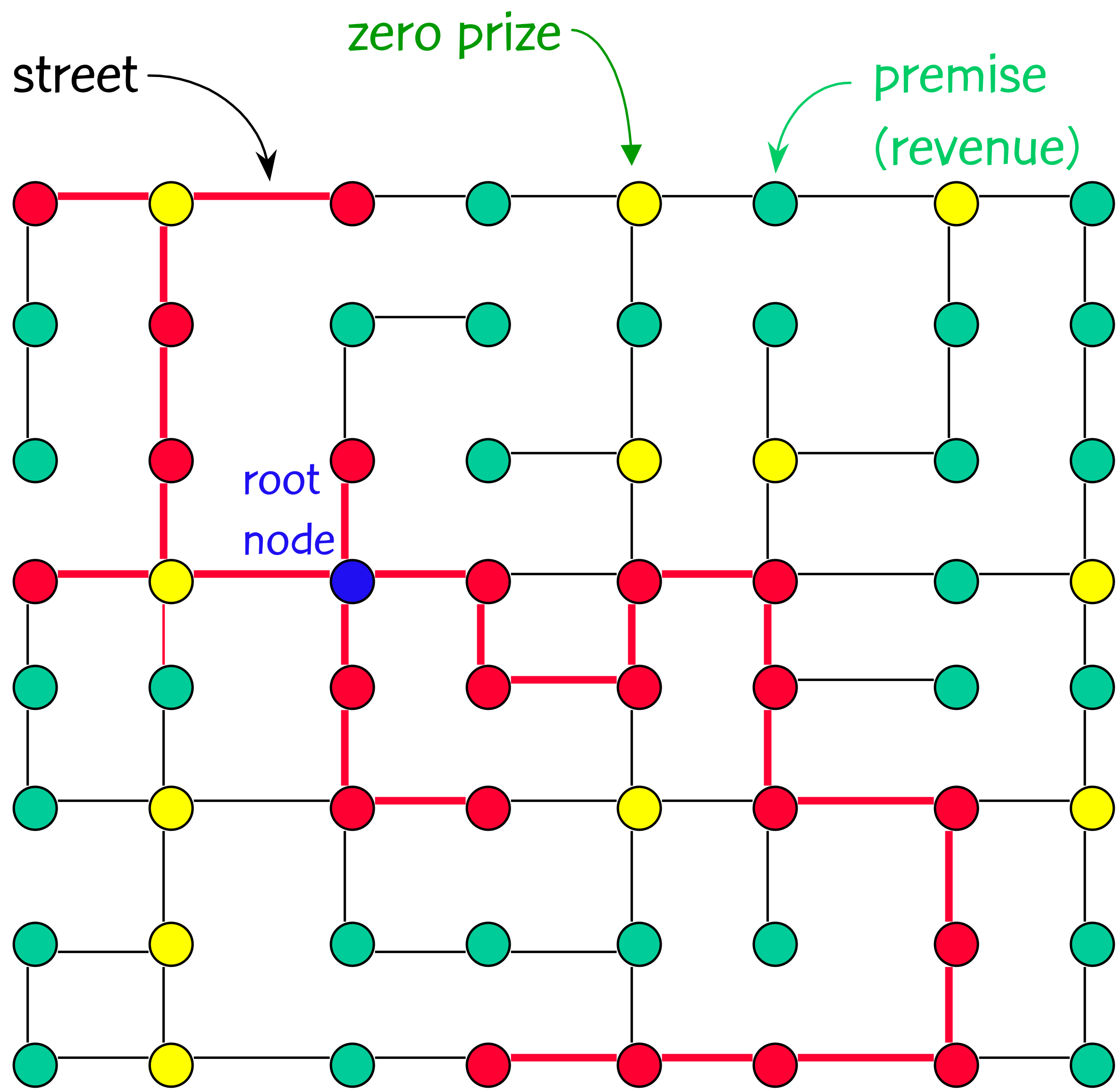
Local access network design



Collect all prizes
(Steiner problem in graphs)



Collect some prizes
(Prize-collecting Steiner Problem in Graphs)



Literature

- Introduced by Bienstock, Goemans, Simchi-Levi, & Williamson (1993)
- Goemans & Williamson (1993, 1996) describe 5/2 and 2 approximation algorithms
- Johnson, Minkoff, & Phillips (1999) describe an implementation of the 2-opt algorithm of Goemans & Williamson (GW)
- Canuto, Resende, & Ribeiro (1999) propose a multi-start heuristic that uses a randomized version of GW
- Lucena & Resende (2000) propose a polyhedral cutting plane algorithm for computing lower bounds

Local search with perturbations

- Summary
  - Generation of initial solution
  - Local search
  - Multi-start strategy
  - Path-relinking associated with multi-start strategy
  - Variable neighborhood search

Generation of initial solution

- Select X, the set of collected nodes
- Connect nodes in X with minimum weight spanning tree $T(X)$
- Recursively remove from $T(X)$ all degree-1 nodes with prize smaller than its incident edge cost $= T_r(X)$
- Basic strategy:

for ($i = 1$ to MAXITR){

select X_i

compute $T(X_i)$ and $T_r(X_i)$

}

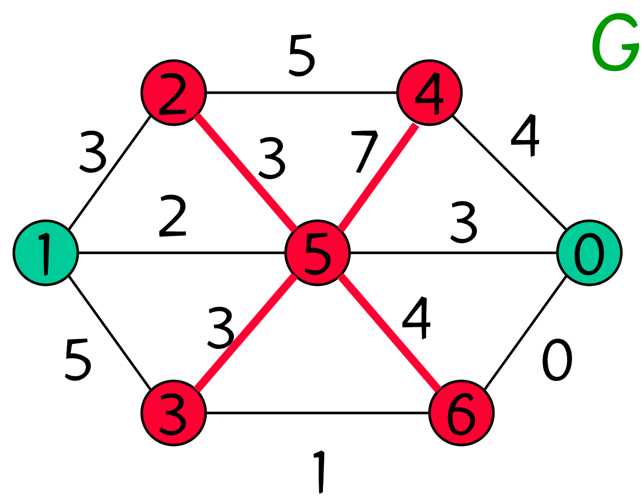
Goemans & Williamson 2-opt algorithm

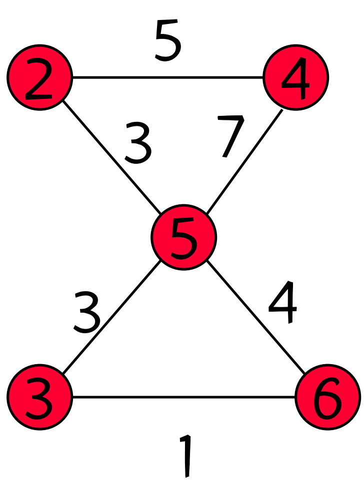
Kruskal's algorithm

Generation of initial solution

Solution obtained by GW: $X = \{2,3,4,5,6\}$

Cost = 18

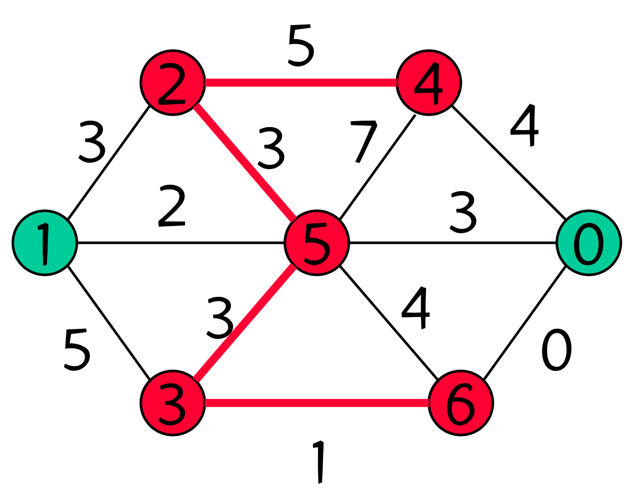




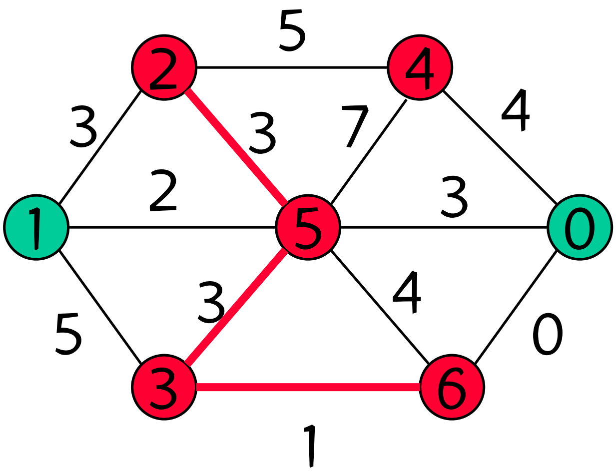
G' = subgraph induced on G by nodes in X

MST solution on G'

Cost = 13



Generation of initial solution

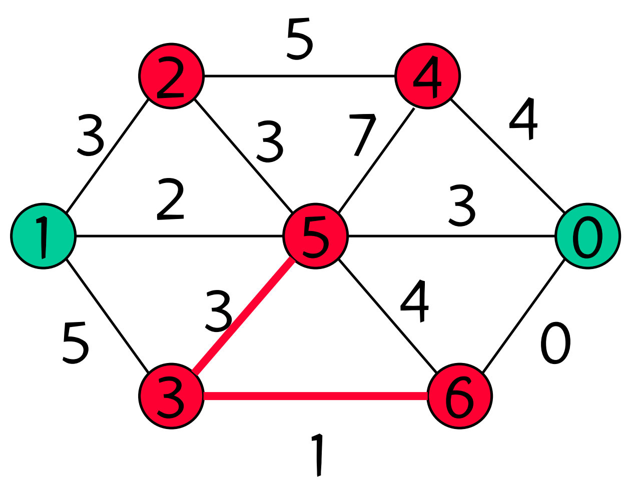


Solution obtained by pruning degree-1 node

Cost = 12

Final solution obtained by pruning another degree-1 node

Cost = 11



Local search

- Representation of solution: set X of vertices in tree $T(X)$
- Neighborhood:
  - $N(X) = \{X' : X$ and X' differ by single node$\}$
  - Moves: insertion & deletion of nodes
- Initial solution: nodes of tree obtained by GW
- Iterative improvement: make move as long as improvement is possible

Local search

improve = TRUE

while (improve){

  improve = FALSE

  circfor i = 1, ..., $|V|$ while .not. improve

  { if ($i \in X$){ $X' = X \setminus \{i\}$}

    else {$X' = X \cup \{i\}$}

    compute tree $T(X')$ and cost(X')

    if (cost(X') < cost(X)){

      $X = X'$

      improve = TRUE

    }

  }

}

if $G(X')$ is disconnected

$\Downarrow$

cost(X') $= \infty$

Multi-start strategy

- Force GW to construct different initial solutions for local search
  - Use original prizes in first iteration
  - Use modified prizes after that
- Modify prizes (two strategies)
  - Introduce noise into prizes

    for $i = 1, \ldots, |V|$ {

    generate $\beta \in [1 - a, 1 + a]$, for $a > 0$

    $d'(i) = d(i) \times \beta$

    }
  - Node elimination
    - Set to zero the prizes of α% of the nodes in nodes(GW) $\cap$ nodes(local search)

Local search with perturbations

```
best = HUGE
d' = d
for ( i = 1, ..., MAXITR ){
    X = GW ( V, E, c, d' )
    X' = LOCALSEARCH( V, E, c, d, X )
    if ( cost(X') < best ){
        X* = X'
    }
    compute perturbations and update d'
}
return X*
```

Approach is similar to a GRASP (greedy randomized adaptive search procedure), in which the greedy randomized construction phase is replaced by the construction with perturbations.

Path relinking

- Integrates intensification & diversification
- Explores the path connecting good solutions
- In local search with perturbations, let
  - X' be the local optimum found by LOCALSEARCH
  - Y be a solution chosen randomly from a POOL of elite solutions
  - $\Delta = \{i \in V : (i \in X'$ and $i \notin Y)$ or $(i \notin X'$ and $i \in Y)\}$
- Construct path between X' (start) and Y (guide):
  - Apply best movement in Δ
  - Verify quality of solution after move
  - Update Δ

Path relinking

- Criteria for inclusion of solution X into POOL of elite solutions
  - If cost(X) is less than smallest cost of POOL solutions
  - Else, if cost(X) is less than largest cost of POOL solutions and X is sufficiently different from all POOL solutions
    - X_1 and X_2 are sufficiently different if they differ by at least β nodes, where β is a fraction of $|V|$

Local search with perturbations and path relinking

POOL $= \phi$

$d' = d$

for ($i = 1, \ldots,$ MAXITR){

    $X =$ GW (V, E, c, d')

    if (X is new){

        $X' =$ LOCALSEARCH(V, E, c, d, X)

        attempt to insert X' into POOL

        select $X'' \in$ RAND(POOL)

        X_{PR} = PATHRELINK(X', X'')

        attempt to insert X_{PR} into POOL

        }

    }

    compute perturbations and update d'

}

return best solution in POOL

Variable neighborhood search

Mladenovic' & Hansen (1997)

- Consider K neighborhoods:
  - $N^1, N^2, \ldots, N^K$
  - $N^k(X) = \{ X' : X$ and X' differ by k nodes$\}$
- Basic scheme (MAXTRY times):
  - Start with initial solution X and $k = 1$
  - while ($k \leq K$){

    choose $X' \in N^k(X)$ at random

    $X' =$ LOCALSEARCH(X') using neighborhood N^1

    $k = k + 1$

    if cost(X') < cost(X) { $X = X'$; $k = 1$}

  }

Local search with perturbations, path relinking, and VNS

```
POOL = ϕ
d' = d
for ( i = 1, ..., MAXITR ){
    X = GW ( V, E, c, d' )
    if ( X is new){
        X' = LOCALSEARCH(V, E, c, d, X )
        attempt to insert X' into POOL
        X'' ∈ RAND(POOL)
        X_PR = PATHRELINK(X', X'' )
        attemp to insert X_PR into POOL
        }
    }
    compute perturbations and update d'
}
X* = best solution in POOL
X* = VNS(V, E, c, d, X* )
return X*
```

Computational results

- 114 test problems
  - From 100 nodes & 284 edges
  - To 1000 nodes & 25,000 edges
  - Three classes:
    - Johnson, Minkoff, & Phillips (1999) P & K problems
    - Steiner C problems (derived from SPG Steiner C test problems in OR-Library)
    - Steiner D problems (derived from SPG Steiner D test problems in OR-Library)

Computational results

- Runs performed on a 400 MHz Pentium II with 32 Mbytes under Linux
- C programming language (gcc)
  - Goemans & Williamson: code of Johnson, Minkoff, and Phillips (1999)
  - Iterative improvement, path relinking, and VNS
- Parameters
  - 500 multi-start iterations
  - Perturbation: $\alpha = 20$ and $a = 1.0$
  - VNS: MAXTRY = 10, $K = 35$
  - Path relinking: $\beta = 0.04\ |V|$ and pool size = 10
  - Alternate between perturbation schemes

Computational results

- Heuristic found:
  - 89 of 104 known optimal values (86%)
  - solution within 1% of lower bound for 104 of 114 problems

Number of optima found with each additional heuristic

type	num	GW	+LS	+PR	+VNS	tot
C	38	6	2	25	3	36
D	32	5	6	10	4	25
JMP	34	8	6	12	2	28
	104					89

Computational results

Number of instances with given relative error

heuristic	< 1%	< 5%	<10%	max (%)
GW	7	22	29	36.4
+LS	17	34	37	11.1
+PR	35	38	40	9.1
+VNS	38	40	40	1.1

Problem type Steiner C

Computational results

Number of instances with given relative error

heuristic	< 1%	< 5%	<10%	max (%)
GW	7	21	31	38.5
+LS	22	33	36	30.8
+PR	34	38	39	10.5
+VNS	34	40	40	4.5

Problem type Steiner D

Computational results

Number of instances with given relative error

heuristic	< 1%	< 5%	<10%	max (%)
GW	15	31	34	6.6
+LS	24	34	34	3.7
+PR	32	34	34	3.4
+VNS	32	34	34	3.4

Problem type JMP

Parallel implementation

- Environment:
  - Cluster with 32 processors: P-II 400
  - Switch IBM 8274 at 10 Mbps
  - LAM 6.3-b3 implementation of MPI
  - Linux and hcc compiler
- Each processor runs a copy of the same program:
  - 200 iterations of LS with perturbations
  - Even-ranked processors: VNS from best solution
  - Odd-ranked processors: randomly select the initial solution for VNS
- Seven additional optimal solutions: 89+7=96 out of 104 optima known

  (series JMP: +2,   C: +2,   D: +3)

Concluding remarks

- Cutting planes algorithm produced tight lower bounds and feasible upper bounds for most instances.
  - Running times were high for most difficult instances (days, even weeks)
- With substantially less computational effort, the heuristic produced optimal and nearly optimal solutions.
  - Running times for most difficult instances averaged about 10,000 seconds
  - Over 90% of solutions were within 1% of lower bound
- Each component contributes to improve the effectiveness of the heuristic

Computational results

lower bounds

- Cutting planes algorithm
  - Found optimal LP solutions in 97 of the 114 test problems (85%)
  - Found tight lower bounds (equal to best known upper bounds) in 104 instances (91%)
  - Of the 97 optimal LP solutions, 94 were integral. Each of the 3 fractional solutions was off of the best known upper bound by less than ½
  - On the 12 instances for which tight lower bounds were not produced, the bounds produced had at most a 1.3% deviation from the best known upper bounds
  - In 13 of the 114 instances, single vertex optima were found
  - In 7 instances the algorithm took over 100,000 seconds to converge to a lower bound. The longest run took over 10 CPU days.