A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs

Celso C. Ribeiro

Catholic University of Rio de Janeiro

Brazil

celso@inf.puc-rio.br

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

Workshop on Algorithm Engineering as a New Paradigm

Kyoto, November 2000

Joint work with E. Uchoa and R. F. Werneck

Outline

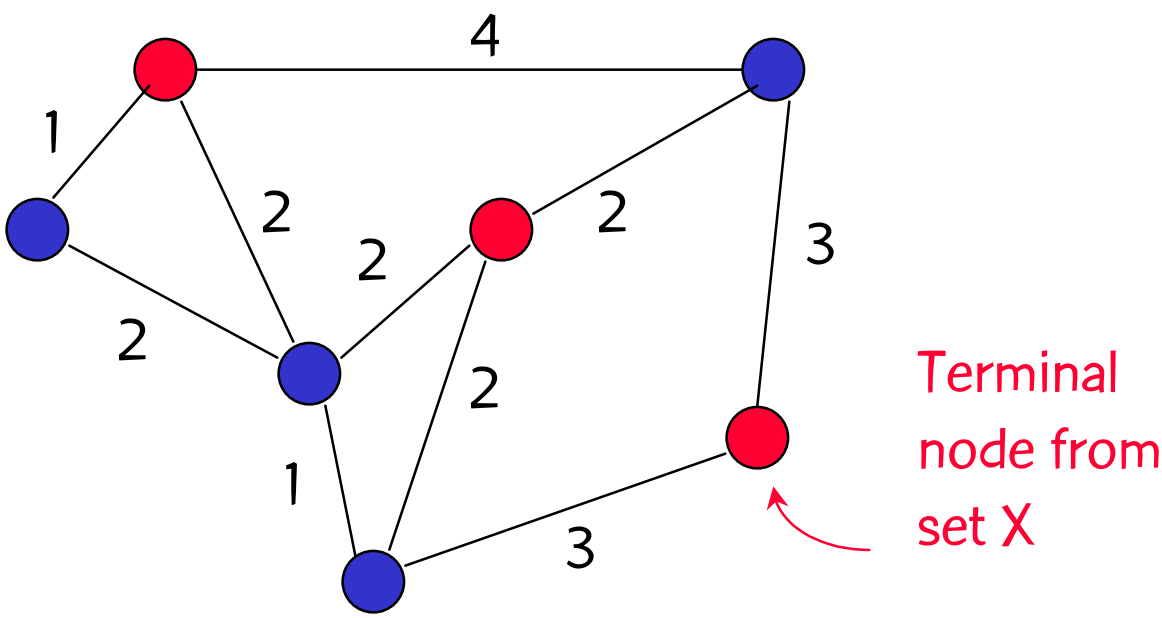
- Steiner problem in graphs
- Test problems and preprocessing
- GRASP
- Hybrid GRASP with perturbations
  - weight perturbation strategy
  - construction heuristics
  - circular local search
  - adaptive path-relinking
- Computational results
- Concluding remarks

Steiner Problem in Graphs: Formulation

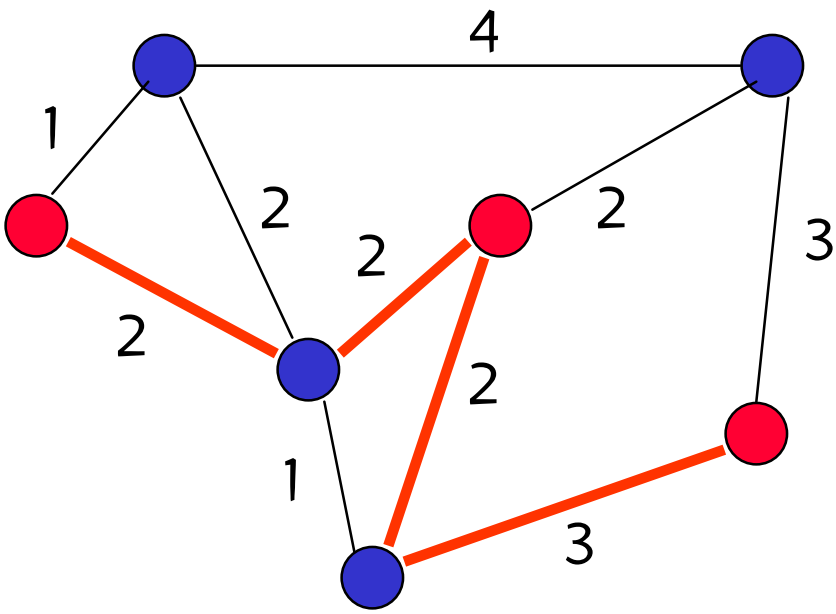
- Given:
  - graph G= (V,E) with n nodes and m edges
  - subset X of terminal nodes
  - edge weights w_{ij}
- SPG: Find a subgraph of G that
  - is connected
  - contains all nodes of X
  - is of minimum weight

Steiner Problem in Graphs: Example

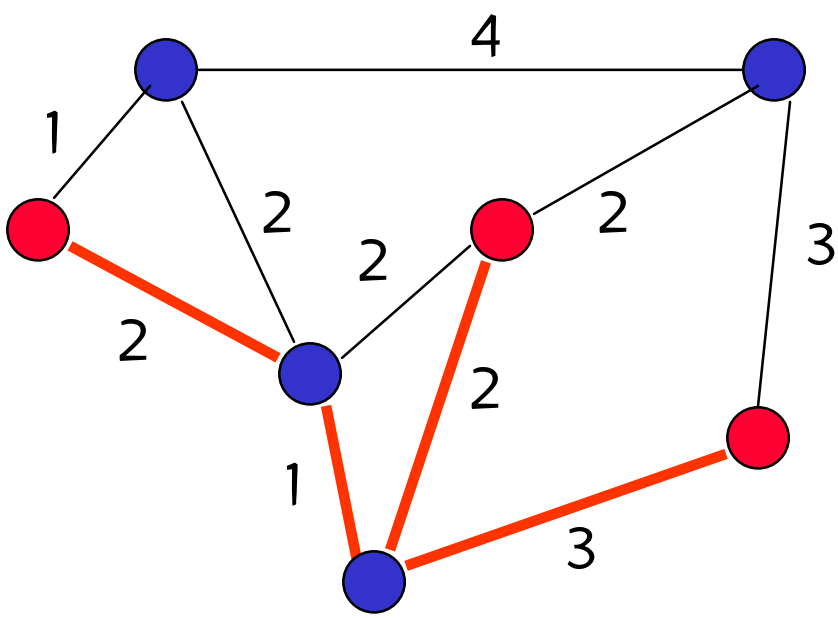
- Classic combinatorial optimization problem (Hwang, Richards & Winter 92)
- Example:



Steiner Problem in Graphs: Example



total weight = 9



total weight = 8
(optimum)

Steiner Problem in Graphs: Complexity

- NP-complete (Karp 72)
- Remains NP-complete for:
  - grid graphs
  - bipartite graphs
  - chordal and split graphs
- Polynomial time algorithms exist for special graphs, e.g.
  - permutation graphs
  - distance hereditary graphs
  - homogeneous graphs

Steiner Problem in Graphs: Applications

- Telecommunications network design
- VLSI design
- Computational biology (phylogenetic trees & DNA codes)
- Reliability
- Examples of applications are found e.g. in the books by Voss (1990) and Hwang, Richards & Winter (1992).

Steiner Problem in Graphs: Algorithms

- Preprocessing
  - Duin 94
  - Koch & Martin 98
  - Uchoa et al. 99
- Exact algorithms
  - Branch-and-cut: Koch & Martin 98
- Construction heuristics
  - Shortest path heuristic: Takahashi & Matsuyama 80
  - Distance network heuristic: Choukhmane 78, Kou et al. 81, Plesník 81+ Mehlhorn 88

Steiner Problem in Graphs: Algorithms

- Local search
  - Node-based: Minoux 90, Voss 90
  - Path-based: Verhoeven et al. 96
  - Hybrid: Martins et al. 99
- Metaheuristics
  - Simulated annealing: Dowsland 91
  - Genetic algorithms: Esbensen 95, Kapsalis et al. 93
  - Tabu: Bastos & Ribeiro 99, Gendreau et al. 99, Ribeiro & Souza 2000
  - GRASP: Martins et al. 98, 99, 2000

Test Problems and Preprocessing

- Wide variety of classes
- OR-Library instances:

  20 problems in each of the series:

  - Series C: |V| = 500 nodes
  - Series D: |V| =1000 nodes
  - Series E: |V| = 2500 nodes

  |E| = 1.25×|V| to 25×|V|

  |X| = 5 to |V|/2

  Significant reductions obtained with preprocessing: reduction tests of Duin and Volgenant 89, plus new tests of Uchoa et al. 99

Test Problems and Preprocessing

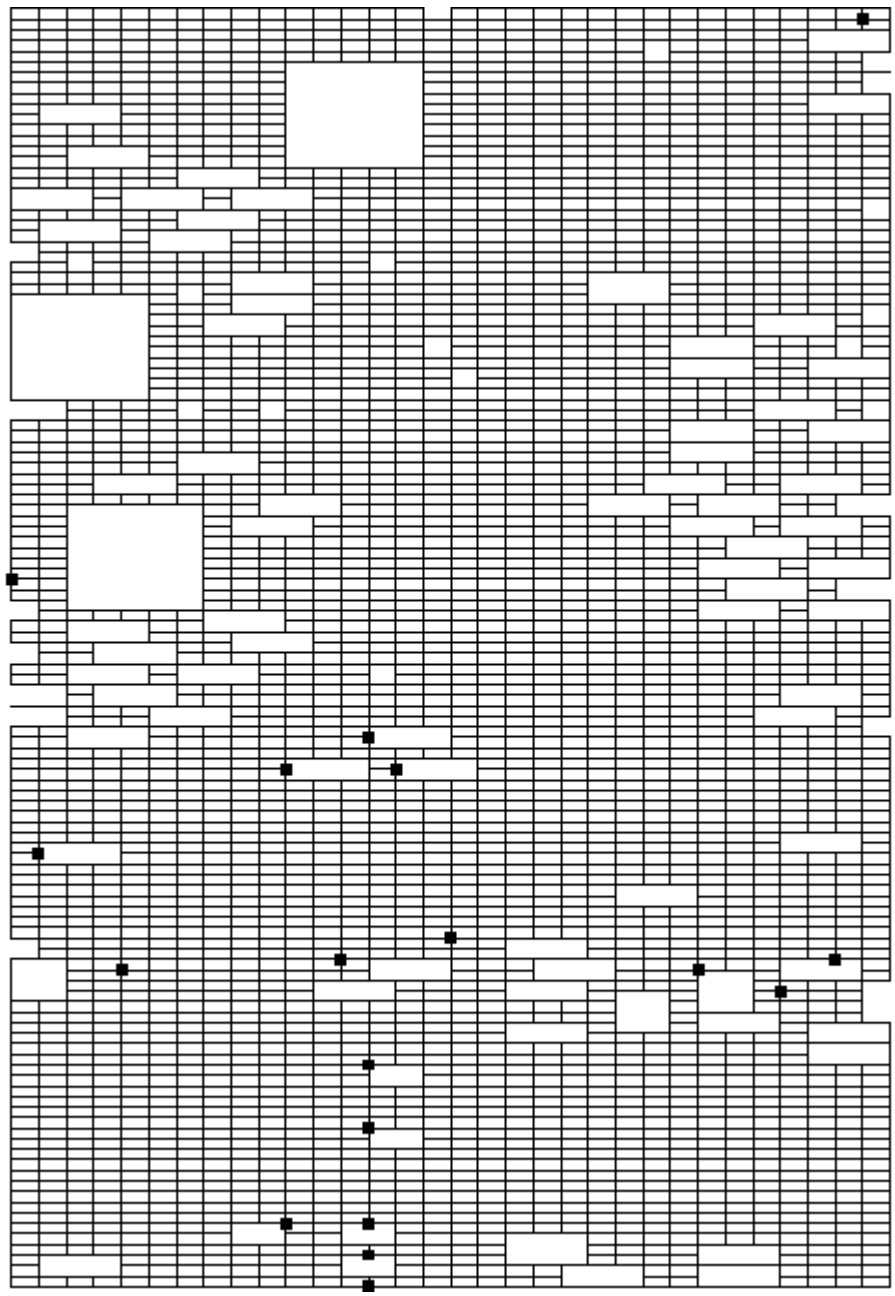
- VLSI instances:
  - 116 instances extracted from real VLSI problems (Koch & Martin 98), defined on grid graphs with holes.
  - Largest problems: |V|=36,711; |E|=68,117; |X|=10 to 2,344

  Koch & Martin 98: 83 optimal solutions

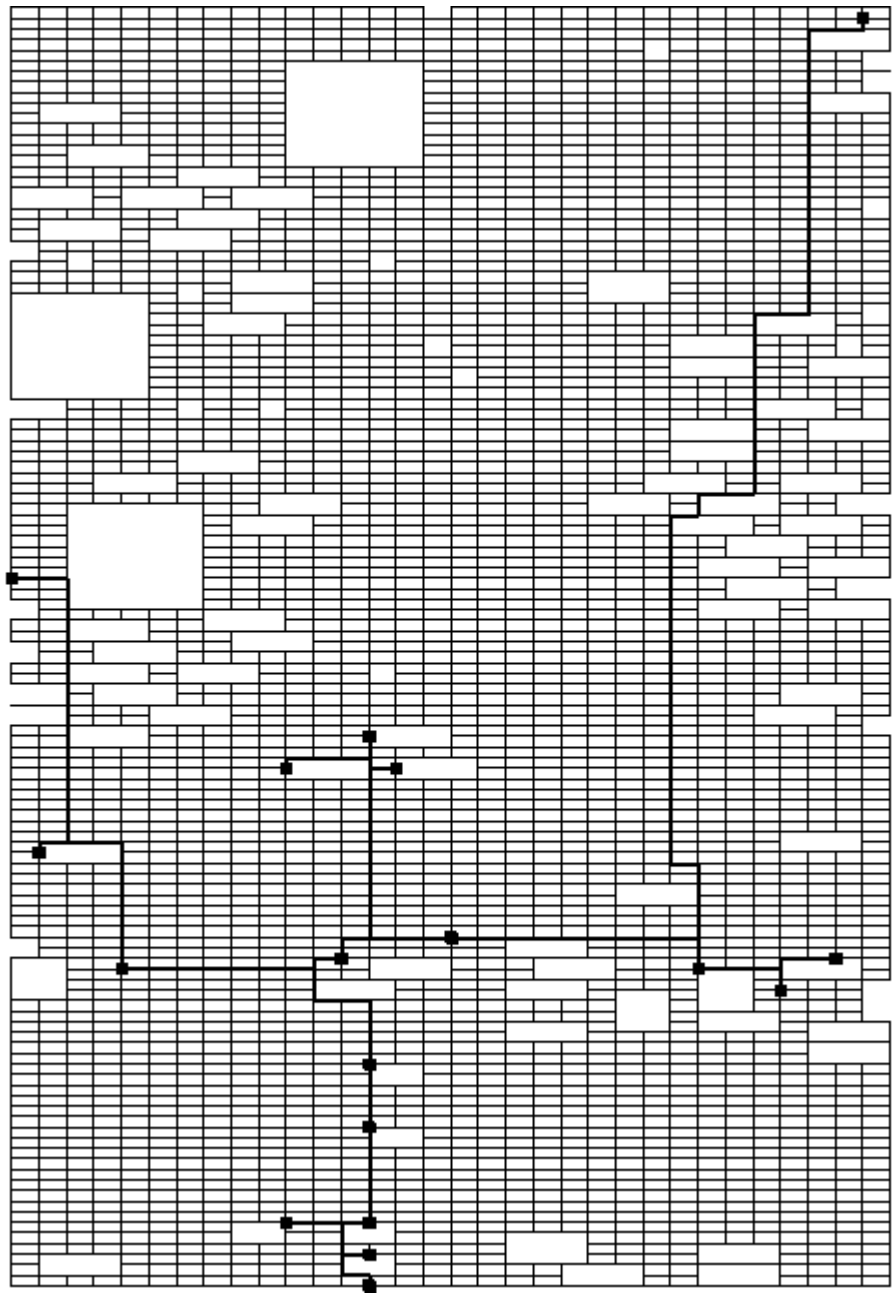
  Uchoa et al. 99: additional 29 optima

  Instances cannot be significantly reduced by traditional reduction tests: new effective tests by Uchoa et al. 99

Test Problems: VLSI Instance diw0559



Test Problems: VLSI Instance diw0559



Test Problems and Preprocessing

- Incidence instances:

  Four series with 100 problems each:

  - Series dv-80: |V|=80 nodes
  - Series dv-160: |V|=160 nodes
  - Series dv-320: |V|=320 nodes
  - Series dv-640: |V|=640 nodes

  Created by Duin & Voss 97, so as to make the reduction tests ineffective.

Test Problems and Preprocessing

- Preprocessing helps the heuristics to find better solutions in smaller times (fixations and reductions).
- Preprocessing can be quite effective: strong reductions
- Some optimal solutions found:
  - OR-Library: C-20, D-20, and E-20
  - 39 VLSI instances
- Remaining instances: 57 reduced OR-Library, 77 reduced VLSI, and 400 original Incidence instances

GRASP: Greedy Randomized Adaptive Search Procedures

- Feo & Resende 89, 95
- Metaheuristic for combinatorial optimization problems
- Multi-start approach
- Each iteration has two phases:
  - Construction
  - Local search
- Seeks to capture good features of greedy algorithms (solution quality and fast local search convergence) and random construction procedures (diversification)

GRASP: Algorithm

```
best_obj = BIG;
repeat many times{
    x = grasp_construction( );
    x = local_search(x);
    if ( obj_function(x) < best_obj ){
        x* = x;
        best_obj = obj_function(x);
    }
}
```

bias towards greediness

good diverse solutions

GRASP: Construction

- repeat until solution is constructed
  - For each candidate element
    - apply a greedy function to element
  - Rank all elements according to their greedy function values
  - Place well-ranked elements in a restricted candidate list (RCL)
  - Select an element from the RCL at random and add it to the solution

GRASP: Local Search

- There is no guarantee that constructed solutions are locally optimal w.r.t. simple neighborhood definitions.
- It is usually beneficial to apply a local search algorithm to find a locally optimal solution.

GRASP: Local Search

- Let
  - N(x) be set of solutions in the neighborhood of solution x.
  - f(x) be the objective function value of solution x.
  - x^0 be an initial feasible solution built by the construction procedure
- Local search (first improvement) to find local minimum

```
while ( there exists y ε N(x) | f(y) < f(x) ){
        x = y;
}
```

Hybrid GRASP with Weight Perturbations

- Replace GRASP construction by a number of greedy heuristics using perturbated weights randomly recomputed at each iteration:
  - More flexibility in algorithm design
  - More effective when greedy algorithm (e.g.shortest path heuristic) not very sensitive to randomization
- Post-optimization (intensification) using path-relinking and elite sol's.
- Successful application to prize-collecting SPG (Canuto et al. 99) using Goemans and Williamson's primal-dual algorithm

Hybrid GRASP with Weight Perturbations

```
read and preprocess input data;
for k = 1,..., max-iterations {
        w(i) = perturbation(w);
        x = greedy_construction(w(i));
        x = local_search(x,w);
        update pool of elite solutions;
}
post-optimization with path-relinking;
return best solution found;
```

Hybrid GRASP: Weight Perturbation Strategy

- Weights randomly perturbated at each iteration i=1,...,max-iterations
- Memory: $t_e^{(i)}$ = number of locally optimal solutions in which edge e appeared after first (i-1) iterations

$$w_e^{(i)} \leftarrow \text{Unif}\left[w_e, r_e^{(i)} \cdot w_e \right]$$

Method	$r_e^{(i)}$
Diversification	$1.25 + 0.75 \cdot t_e^{(i)} / (i-1)$
Intensification	$2 - 0.75 \cdot t_e^{(i)} / (i-1)$
Uniform	2

Weight Perturbation Strategy: Results

- 370 instances (57 OR-Library, 73 VLSI for which the optimum is known, 240 Incidence instances with up to 320 nodes)
- 128 hybrid GRASP iterations followed by adaptive path-relinking
- Each instance run five times
- Seven perturbation strategies combining methods I, D, and U
- Discarded instances for which all perturbation strategies led to the same average solution value over the five runs (151 remained)

Weight Perturbation Strategy: Results

- Evaluation criteria:
  - For each run: relative error (%) with respect to the best average solution value among those found by the seven strategies (average over all instances; smallest is best)
  - For each instance: score of some method M is the number of methods which found better average solution values than M (sum over all instances; smallest is best)
  - Best: number of instances for which some method led to the best average solution value (largest is best)

Weight Perturbation Strategy: Results

Strategy	Error (%)	Score	Best
D	0.099	273	55
DU	0.106	277	45
I	0.136	403	30
ID	0.110	351	38
IDU	0.096	**250**	**56**
IU	**0.077**	251	55
U	0.106	296	46

- IDU: most robust perturbation strategy, leading to best overall results

Hybrid GRASP: Construction Heuristics

- One of a number of construction heuristics is randomly chosen at each iteration i=1,...,max-iterations
- Heuristic 1: SPH
  - Randomly select any terminal as root.
  - At each iteration, let H be the already spanned terminals and $s \in X \setminus H$ be the closest non-spanned terminal.
  - Append to current tree all nodes in the shortest path from s to it.
  - Set $H \leftarrow H \cup \{s\}$ and stop when H=X

  Complexity: $O(|X| \cdot |E| \cdot \log |V|)$

Hybrid GRASP: Construction Heuristics

- Heuristic 2: Kruskal-based
  - Initially, $|X|$ isolated components, each made up by a single terminal.
  - At each iteration, find the shortest path conecting two components and merge them together with this path.
  - Stop after all components have collapsed into a single component.

  Complexity: $O(|X| \cdot |E| \cdot \log|V|)$

Hybrid GRASP: Construction Heuristics

- Heuristic 3: MST-based
  - Initially, compute a minimum spanning tree of the original graph G.
  - If all leaves are terminals, then stop and return the current tree.
  - Otherwise, remove all non-terminal leaves from the current tree.
  - Compute a minimum spanning tree of the graph induced in G by the nodes remaining in the tree after pruning.
  - Repeat the steps above.

  Complexity: $O(|X| \cdot |E| \cdot \alpha(|E|,|V|))$

  (can also be applied with the others)

Hybrid GRASP: Construction Heuristics

- Other heuristics could also be used
- Combination of different heuristics may generate structurally different solutions, driving the search to different regions.
- More robust algorithm, since a single heuristic is not likely to be appropriate to all classes of instances.
- More heuristics will deal more efficiently with a wider variety of classes of problem instances.

Hybrid GRASP: Construction Heuristics

- First three hybrid GRASP iterations:
  - Original weights (no perturbations)
  - Apply each heuristic exactly once.
- Forthcoming iterations:
  - Use weight perturbation method I, next method D, then method U, then method I again, and so on.
  - Randomly select one of the construction heuristics.
- Alternance between intensification and diversification: strategic oscillation approach

Construction Heuristics: Computational Results

- Same 370 instances already used
- 128 hybrid GRASP iterations followed by adaptive path-relinking
- Each instance run five times
- Four combinations of construction heuristics 1 (T), 2 (K), and 3 (M):
  - SPH always included (fastest and best individual results)
- Discarded instances for which all combinations led to the same average solution value over the five runs (113 remained)

Construction Heuristics: Computational Results

Strategy	Error (%)	Score	Best
T	**0.091**	153	41
TK	0.092	144	45
TM	0.109	163	33
TMK	0.097	**125**	**46**

- Same evaluation criteria: relative error (%), score, best solutions found
- TMK: most robust combination strategy, leading to best overall results

Hybrid GRASP: Local Search

- Start from solution obtained at the end of the construction phase
- Neighborhood definitions:
  - Node-based neighborhood
  - Path-based neighborhood
- Hybrid local search (Martins et al. 99):
  - Use original weights.
  - Start local search using one of the above neighborhoods.
  - Use the other neighborhood after a local optimum is found.
  - Alternate until the current solution cannot be further improved.

Local Search: Node-based Neighborhood

- Associate a solution (Steiner tree) with each subset $S \subseteq V \backslash X$ of Steiner nodes such that the graph induced in G by $S \cup X$ is connected.
- S^*: set of Steiner nodes in the minimum weighted Steiner tree
- Optimal Steiner tree is the minimum spanning tree of the graph induced in G by $S^* \cup X$.
- Node-based neighborhood $N_n(S)$:
  - Add to S a new non-terminal node s.
  - Remove a Steiner node s from S.

Local Search: Node-based Neighborhood

- Each solution S has at most $|V|-|X|$ feasible neighbors.
- Nodes examined for insertion or elimination in circular order.
- Insertions:
  - Check feasibility in $O(|V|)$ time.
  - Use Kruskal's algorithm to recompute MST in $O(|V|.\alpha(|V|,|V|))$ time.
- Eliminations:
  - Use Kruskal's algorithm to recompute MST in $O(|E|.\alpha(|E|,|V|))$ time.
- Prune non-terminal nodes with degree one

Local Search: Node-based Neighborhood

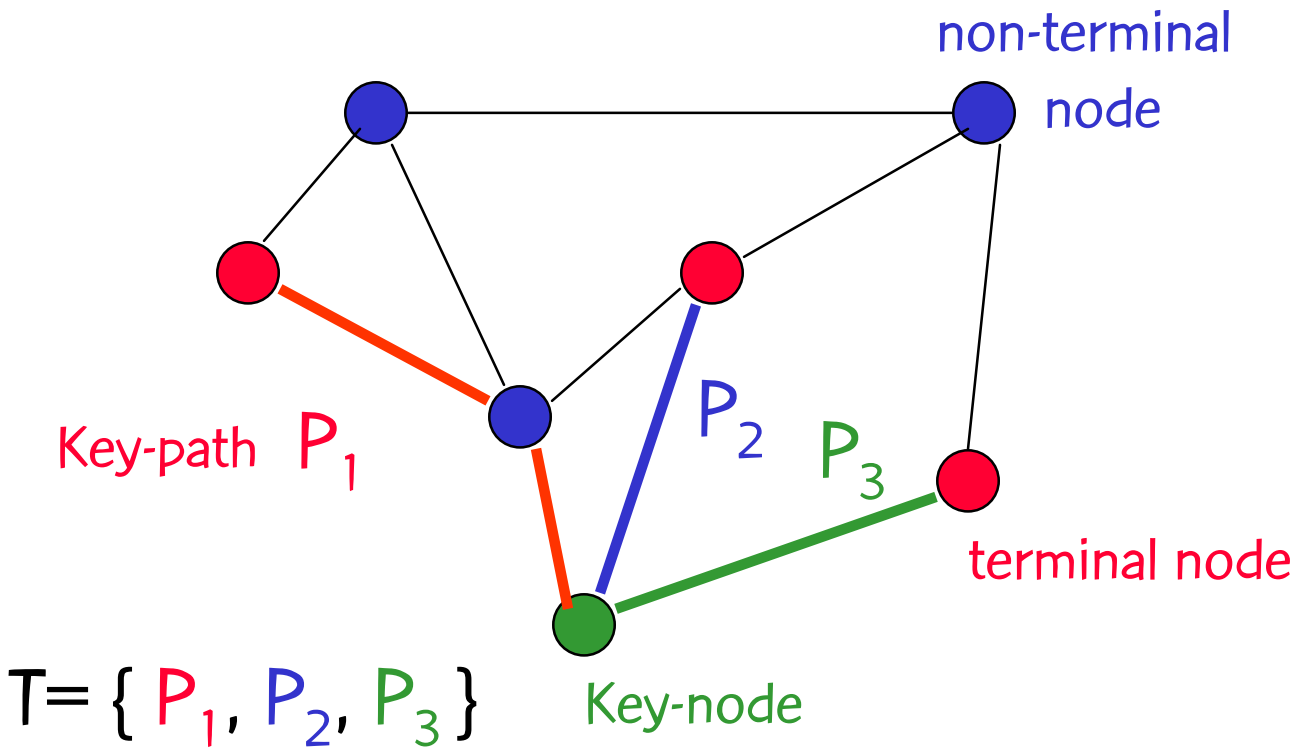
- First improving strategy: feasible neighbor replaces current solution whenever its cost is not greater than that of the latter.
- Local search resumes from the next to be examined node.
- To drive different applications of the local search to different solutions, at each iteration the nodes are investigated in a circular order defined by a different random permutation of their original indices.

Local Search: Path-based Neighborhood

- Proposed by Verhoeven et al. 96
- Key-node: Steiner node with degree ≥ 3 (at most $|X|-2$)
- Key-path: path in a Steiner tree connecting terminals or key-nodes in which all intermediate nodes are Steiner nodes with degree two (at most $2.|X|-3$).

Local Search: Path-based Neighborhood

- A minimum Steiner tree consists of key-paths that are shortest paths between key-nodes or terminals.



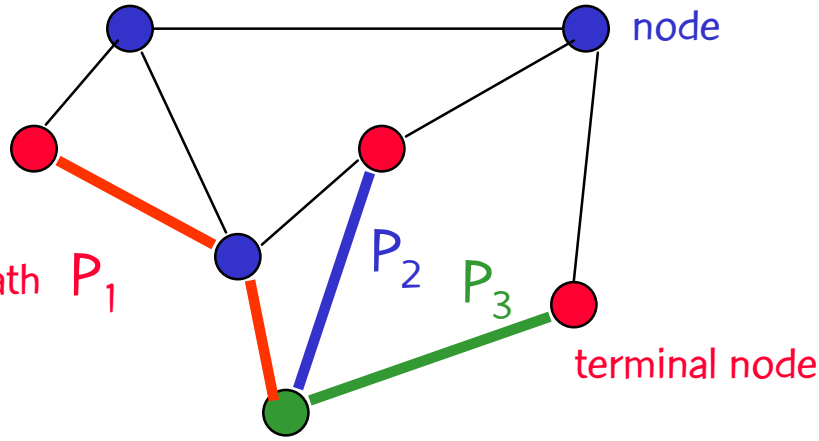
$T = \{ P_1, P_2, P_3 \}$

Local Search: Path-based Neighborhood

- Path-based neighborhood $N_p(T)$:
  - Replace a key-path by the shortest path connecting the two components resulting from its removal.
- Each solution T has at most $2.|X|-3$ neighbors.
- Solutions only have neighbors with lower or equal cost.
- Local minima have no neighbors: neighborhood is not connected.

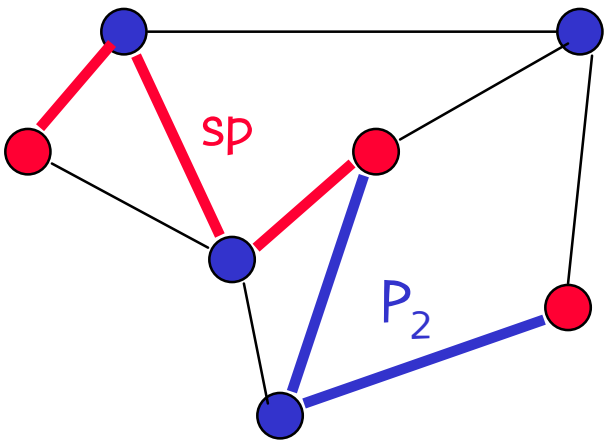
Neighborhood

T

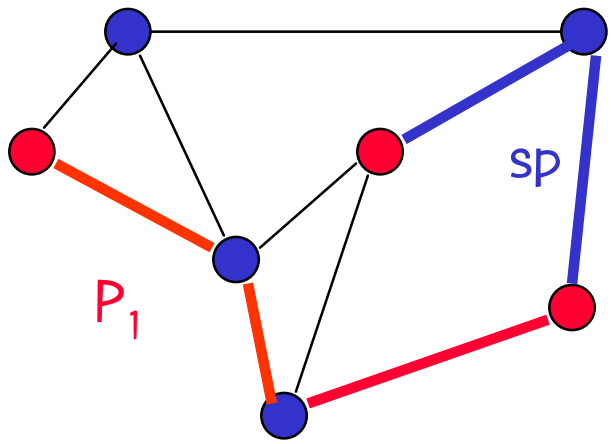


$T = \{ P_1, P_2, P_3 \}$

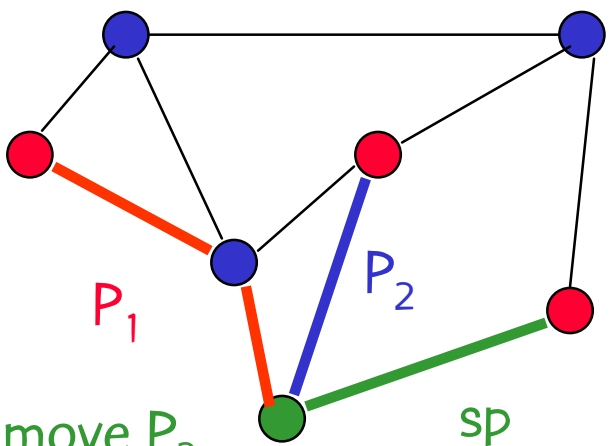
Neighbor 1: remove P_1



Neighbor 2: remove P_2



N(T)



Neighbor 3: remove P_3

Local Search: Path-based Neighborhood

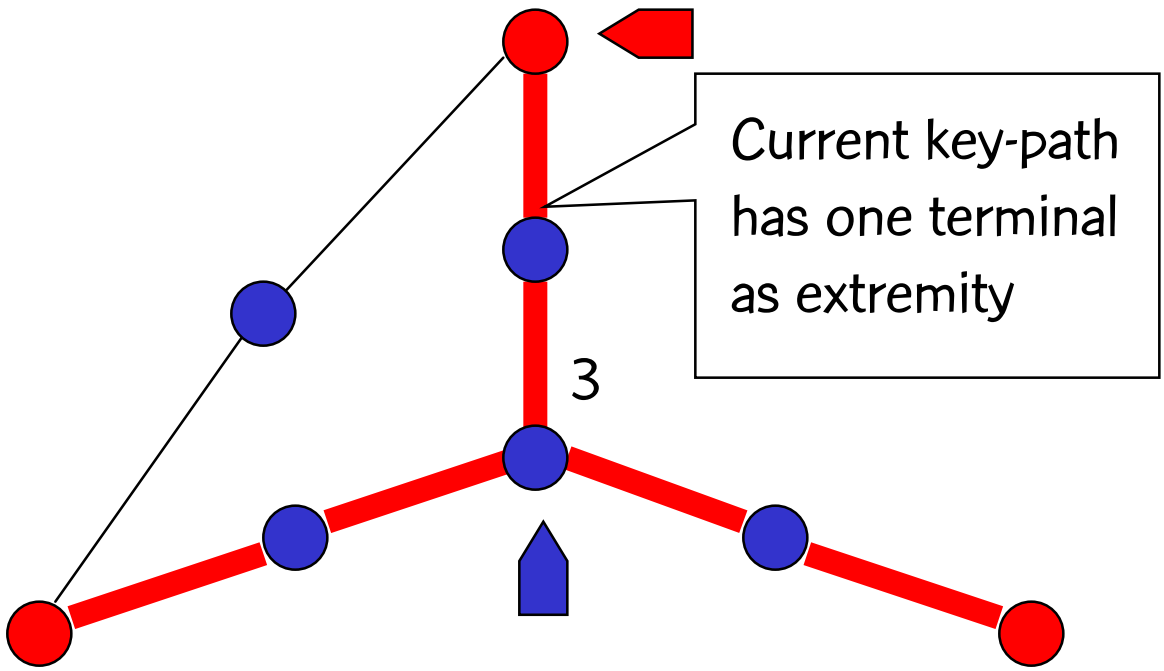
- At each iteration, nodes are examined in a different circular order (defined by a different random permutation of their original indices) and all key-paths originating at each node are investigated (avoiding repetitions).
- Search always move to the first improving neighbor and stops after a full pass of all nodes without improvement in the weight of the best solution.

Local Search: Path-based Neighborhood

- Criteria for determining whether a neighbor is improving or not:
  - (a) new solution has strictly smaller weight: ultimate goal of the search

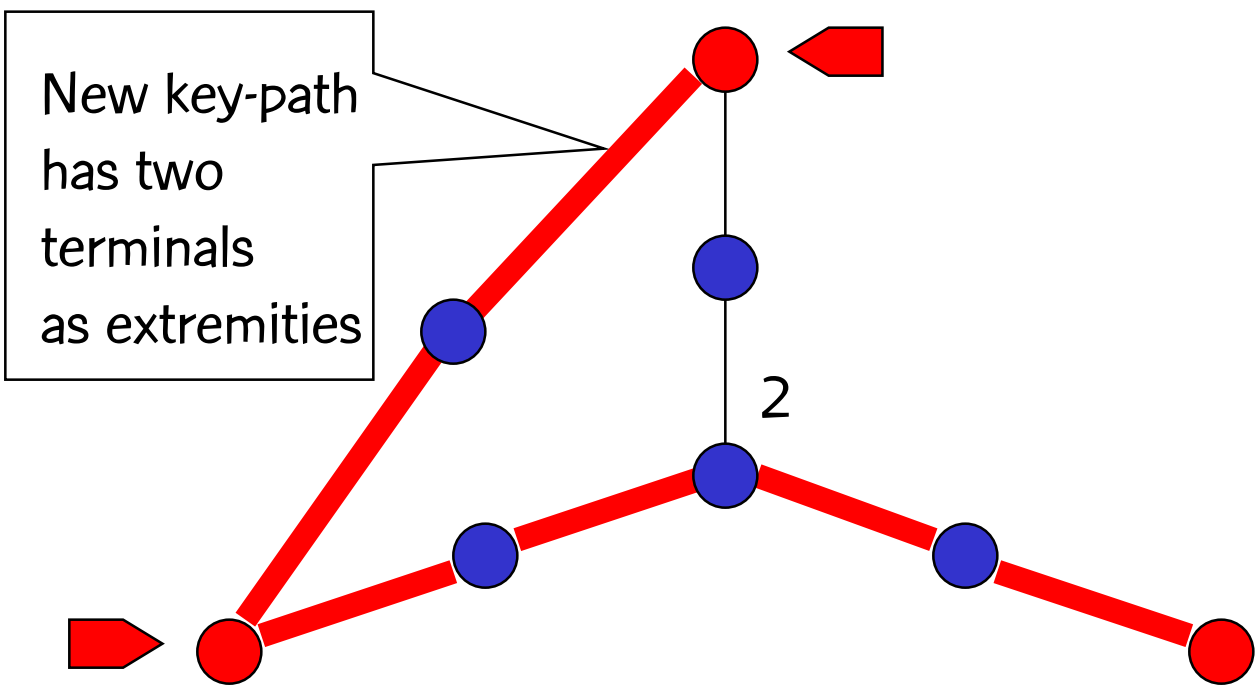
Local Search: Path-based Neighborhood

- (b) new key-path has more terminals as extremities: leads to the reduction of the degree of at least one key-node, then to longer key-paths that will have a greater chance to be removed at some future iteration.



Local Search: Path-based Neighborhood

- (b) new key-path has more terminals as extremities: leads to the reduction of the degree of at least one key-node, then to longer key-paths that will have a greater chance to be removed at some future iteration.



Local Search: Path-based Neighborhood

- (c) new key-path is longer in terms of nodes: more likely to lead to a reduction in solution weight at future iterations.

- These criteria are embedded in the shortest path algorithm itself and are quite effective whenever there are multiple paths connecting the same pair of nodes, which happens very often in "degenerate" instances in which many edges have the same weight (e.g. VLSI instances).

Local Search: Computational Results

- All 530 instances not solved to optimality after preprocessing
- Apply each local search strategy to the solution built by SPH:
  - Node-based neighborhood (N)
  - Path-based neighborhood (P)
  - Hybrid strategy starting with one of the above (NP and PN)
- Discarded instances for which none of the local search strategies have been able to improve the initial solution (454 remained)

Local Search: Computational Results

- Evaluation criteria:
  - Average improvement with respect to the initial solution (average over all instances; largest is best)
  - For each instance: score of some strategy S is the number of strategies which found better average solution values than S (sum over all instances; smallest is best)
  - Best: number of instances for which some strategy led to the best solution value (largest is best)
  - Relative time with respect to N

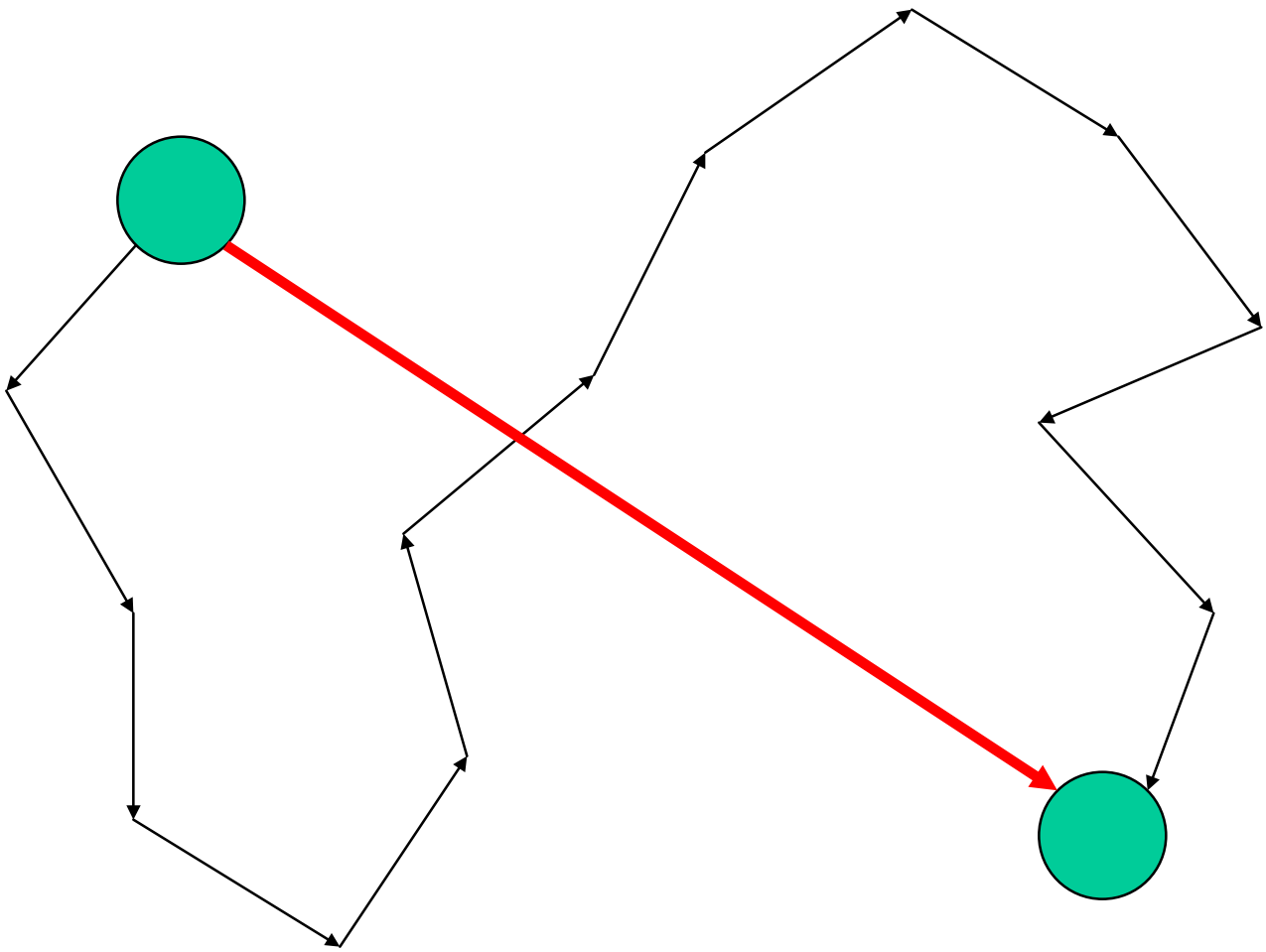
Local Search: Computational Results

Strategy	Impr. (%)	Score	Best	Time
N	9.538	613	170	1.
P	2.591	768	147	1.380
NP	**10.245**	**200**	308	1.381
PN	10.239	254	**321**	2.562

- Path-based consistently faster (same complexity, fewer neighbors)
- Node-based: ineffective for sparse instances (such as VLSI and OR-Library), better for Incidence
- Circular hybrid local search: more effective, not much slower
- First neighborhood is randomly selected at each iteration.

Hybrid GRASP: Path-relinking

- Path-relinking (PR) generates new solutions by exploring trajectories which connect elite solutions.



Hybrid GRASP: Path-relinking

- Starting from elite solutions, PR explores paths leading towards other guiding elite solutions, in the search for better solutions.
- PR favors the selection of moves that introduce attributes contained in the guiding solutions.
- Algorithm handles a pool with a fixed number of at most p elite solutions found along the search.

Path-Relinking: Pool of Elite Solutions

- Pool initialized with p null solutions
- Every solution found at the end of the local search replaces the worst in the pool if it is different from all others and strictly better than the worst one: first (or current) generation of elite solurions.
- Solutions generated by path-relinking are progressively inserted into the next generation pool if the same criteria are satisfied.
- Repeat if best solution was improved, othewise stop.

Path-Relinking: Complementary Moves

- For each pair of elite solutions in the pool: compute their symmetric difference (Steiner nodes which appear in one of them, but not in the other).
- This set defines the list of moves to be aplied to the initial solution.
- Always perform the best remaining move in the list until the guiding solution is attained.
- Best solution found along this trajectory is candidate for insertion into the next generation pool.

Path-Relinking: Weight Penalizations

- For each pair of elite solutions in the pool: weights of edges in only one of them are multiplied by $\alpha \in$ Unif[50,100], those of edges in both of them are multiplied by 2000.
- Apply SPH to graph with new edge weights: new solution is likely to preserve characteristics (edges) shared by both solutions.
- Restore original weights and apply local search to this new solution.
- New solution is tested for insertion in the next generation pool.

Path-Relinking: Adaptive Strategy

- WP scheme is likely to be faster for sparse instances (such as VLSI) with too many non-terminal nodes.
- More effective and robust adaptive strategy.
- First pass after the construction of the first generation pool: apply both schemes to relink the best solution in the pool with every other elite solution also in the pool.
- Then, select and use the scheme with the smallest sampled average computation time.

Path-Relinking: Computational Results

- Same 370 instances already used to study the perturbation strategies
- 128 hybrid GRASP iterations followed by adaptive path-relinking
- Each instance run five times
- Four path-relinking schemes: WR, CM1 (one-way), CM2 (two-way), HA (hybrid adaptive).
- Discarded instances for which none of the path-relinking schemes improved the solution found by the hybrid GRASP (95 remained)

Path-Relinking: Computational Results

Strategy	Impr. (%)	Score	Best	Time
WR	0.286	144	37	3.816
CM1	0.339	111	36	1.
CM2	**0.368**	**55**	**59**	1.944
HA	0.356	**55**	54	1.455

- Same evaluation criteria: relative improvement (%), score, best solutions found, average relative time (with respect to CM1)
- CM2 takes approximately twice as much time as CM1, but is only marginally better.
- CM1 more effective when it starts from the best solution in the pair.

Path-Relinking: Computational Results

Strategy	Impr. (%)	Score	Best	Time
WR	0.286	144	37	3.816
CM1	0.339	111	36	1.
CM2	**0.368**	**55**	**59**	1.944
HA	0.356	**55**	54	1.455

- CM1 faster than WR, but...
- ... although CM1 led to better solutions for Incidence instances, WR was much more effective for OR-Library and VLSI instances.
- Adaptive path-relinking aims at selecting the fastest strategy, but also seems to find the best results.

Path-Relinking: Computational Results

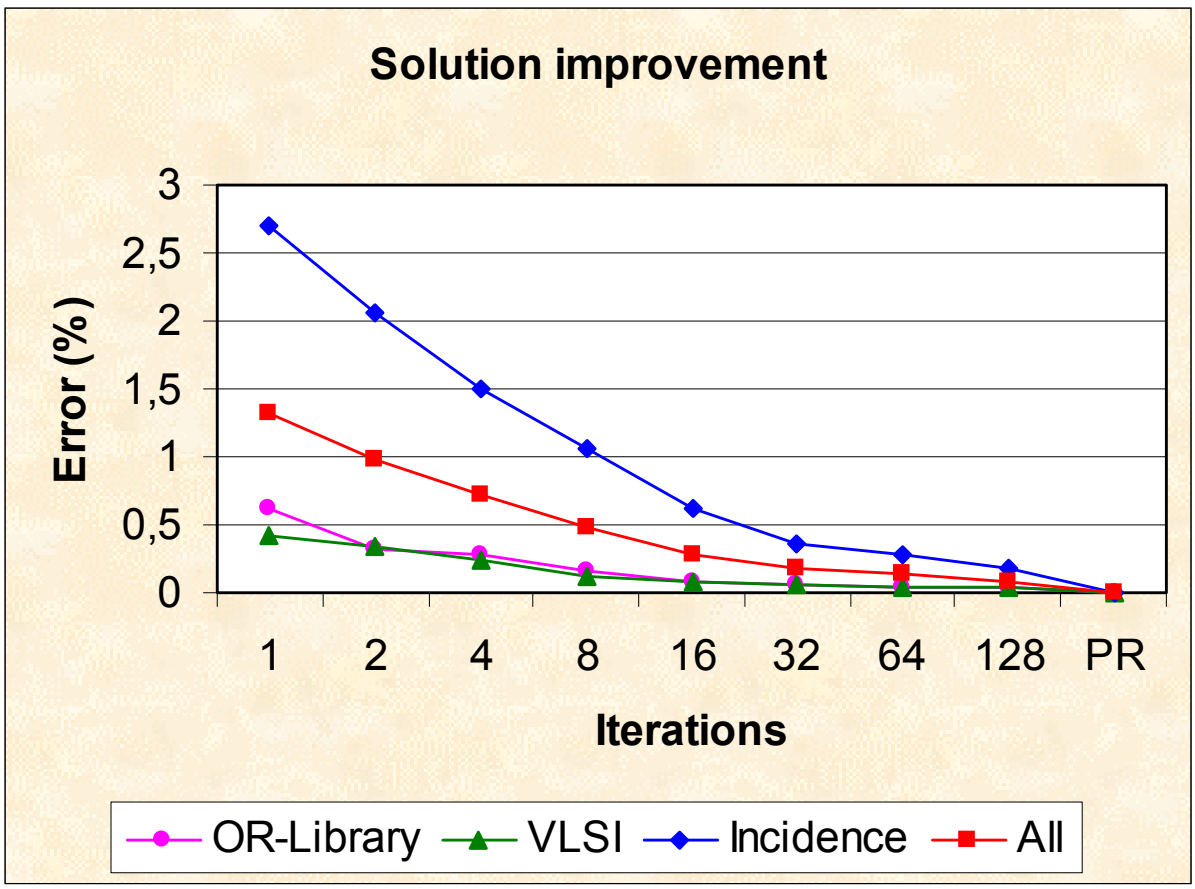
Series (instances)	WR (%)	CM1 (%)
OR-Library (57)	20.4	79.6
VLSI (73)	59.7	40.3
Incidence (240)	6.3	93.7
All (370)	19.0	81.0

- Choices made by the adaptive path-relinking algorithm along 5 runs of each instance (1850 runs).
- Selection made by HA is indeed that leading to smallest times: e.g. WR takes three times as much time as CM1 for OR-Library instances.
- Path-relinking is quite effective.

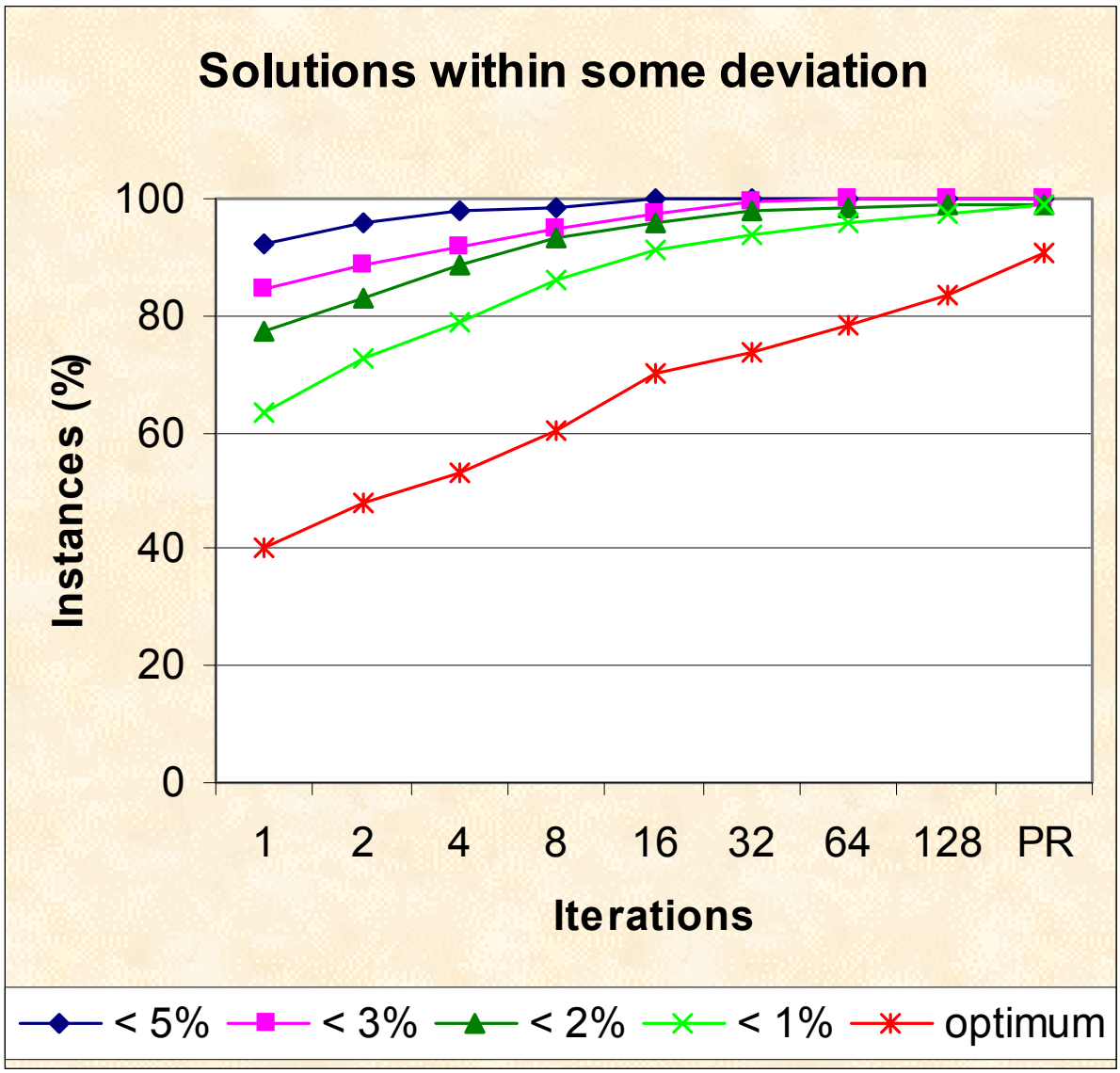
Computational Results

- HGP-PR applied to all instances
- Optimal solutions for most VLSI instances
- Improved best solution known for the four VLSI instances not solved to optimality by Koch & Martin 98.
- Figures:
  - Progress in deviation with respect to the solution found after path-relinking
  - Progress in solution quality with respect to the optimal value

Solution Improvement with Respect to Best Value



Instances within Some Deviation from Optimum



Improvement in Solution Quality

- Solution quality steadily improves with the increase in the number of hybrid GRASP iterations.
- Approximately 90% of all instances within 0.5% from optimality after only 16 iterations
- More iterations are particularly helpful for the larger, most difficult instances.
- Effectiveness of path-relinking

Computational Results: Parameters

- Parameters with influence in solution quality:
  - number of iterations
  - number of elite solutions in the pool
- Further computational experiments:
  - without PR, double the number of iterations (from 128 to 256)
  - same 128 iterations, double the number of elite solutions in the pool (from 10 to 20)
- Selected 15 instances among the hardest ones

Parameters: Iterations

- Improved solutions for four out of 15 selected instances
- Effectiveness of path-relinking: can find even better solutions in less total time than with doubling the number of iterations

	128 iterations		256 iterations	
Instance	HGP	sec	HGP	sec
e18	570	519.9	570	1051.0
alut2288	3846	57.9	3846	114.5
alut2610	12304	1764.6	**12287**	4421.8
taq0014	5335	124.5	5335	261.2
taq0903	5108	148.5	**5107**	324.8

Parameters: Pool Size

- Improved solutions for ten out of 15 selected instances
- Increase of pool size does not necessarily lead to increase in computation time
- Effectiveness of hybrid GRASP can be further increased if more computation time is allowed

	Size = 10		Size = 20	
Instance	HGP-PR	sec	HGP-PR	sec
e18	569	999.2	**567**	948.1
alut2288	3846	91.2	3846	84.0
alut2610	12284	2939.5	**12269**	6524.5
taq0014	5335	175.7	**5326**	236.3
taq0903	5102	463.9	**5099**	485.8

Concluding Remarks

- Hybrid GRASP with perturbations and adaptive path-relinking
- Several features of different metaheuristics:
  - memory-based construction
  - strategic oscillation driven by weight perturbations
  - variable neighborhoods
  - evolutionary population methods
  - path-relinking
  - randomness

Concluding Remarks

- Computational study involving a very broad set of test problems
- HGP-PR outperformed other heuristics:
  - RTS-PR: reactive tabu search with PR (Bastos & Ribeiro 99)
  - TS: tabu search (Ribeiro & Souza 2000)
  - F-Tabu: tabu search (Gendreau et al. 99)
  - SV: vertex-exchange (Duin & Voss 97)

Concluding Remarks

- Average relative errors
- VLSI and dv-640 not systematically addressed in the literature
- Optimal solutions not known for all dv-320 instances (actual errors may be even smaller)
- OR-Library instances: HGP-PR outperformed other heuristics
- VLSI instances: HGP-PR performed remarkably well (improved solutions)
- Incidence instances: HGP-PR and RTS-PR found same quality solutions

Concluding Remarks

- Average relative errors (%) with respect to the optimal values (lower bounds for some dv-320 instances)

Series	HGP-PR	RTS-PR	TS	F-Tabu	SV
C	**0.00**	0.01	0.26	0.02	–
D	**0.04**	0.10	0.71	0.11	–
E	**0.05**	0.17	0.83	0.31	–
OR-Library	**0.03**	0.09	0.60	0.15	–
dv-80	**0.01**	**0.01**	0.08	–	1.03
dv-160	**0.13**	**0.12**	0.35	–	0.98
dv-320	**0.35**	**0.34**	0.89	–	1.20
Incidence	**0.17**	**0.16**	0.44	–	1.07

Concluding Remarks

- Substitution of the greedy randomized construction phase of a GRASP by a greedy construction using weight perturbations already led to sound computational results for the prize-collecting SPG (Canuto et al. 99).
- Use of memory-based construction procedures can strongly improve memoryless approaches (Glover & Fleurent 99)

Concluding Remarks

- Weight perturbations can be used to implement effectively a wide variety of algorithm elements, such as randomized greedy heuristics; intensification and diversification strategies; and path-relinking.
- Effectiveness of path-relinking to improve solutions found by metaheuristics such as tabu search and GRASP
- PR also plays a major role in terms of the robustness of the heuristic.