A parallel GRASP for the Steiner problem in graphs using a hybrid local search

Maurício G. C. Resende
Algorithms & Optimization Research Dept.
AT&T Labs Research
Florham Park, New Jersey

mgcr@research.att.com
http://www.research.att.com/~mgcr

INFORMS Philadelphia Meeting
November 1999

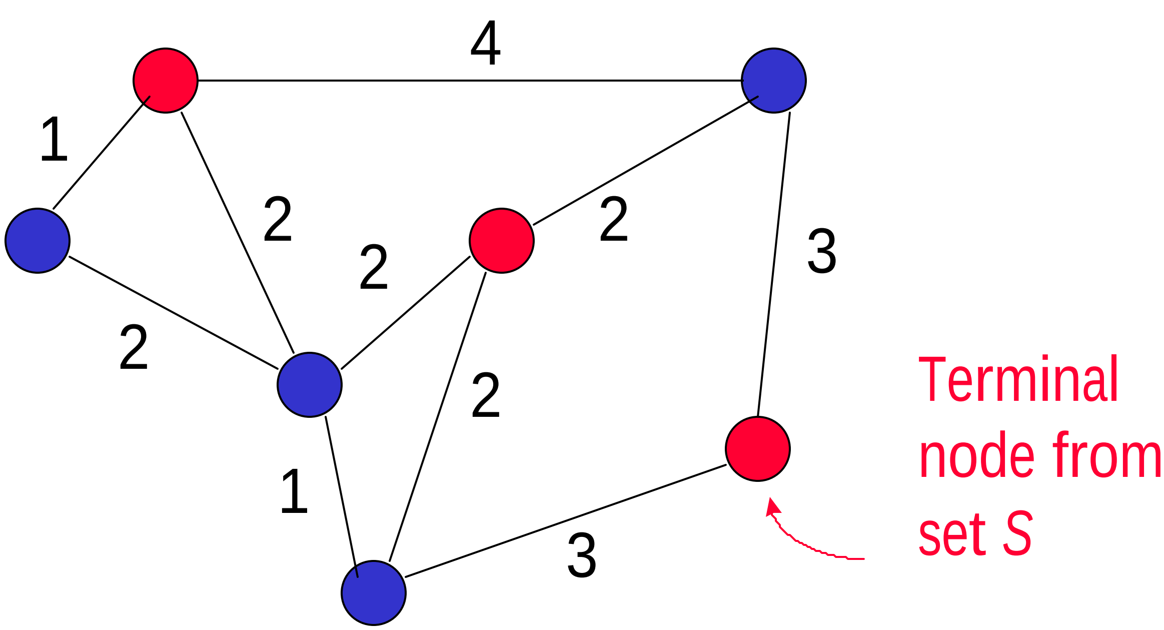
Joint work with S. Martins, C. C. Ribeiro, and P. M. Pardalos

Steiner Problem in Graphs (SPG)

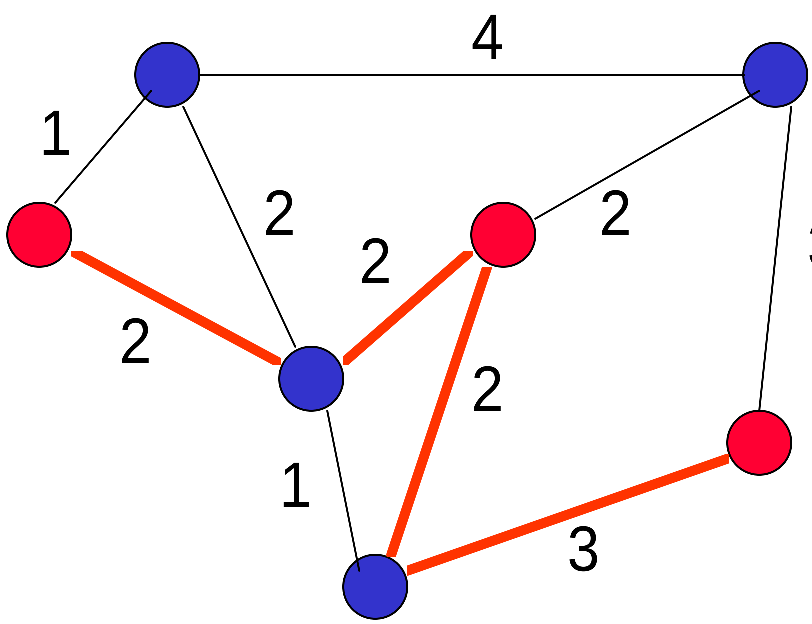
- Given
  - a graph G (V,E) with n vertices and m edges
  - a subset S of the vertices V
  - edge weights $w_1, w_1, ..., w_m$
- SPG: Find a subgraph of G that
  - is connected
  - contains all vertices of S
  - is of minimum weight

Steiner Problem in Graphs

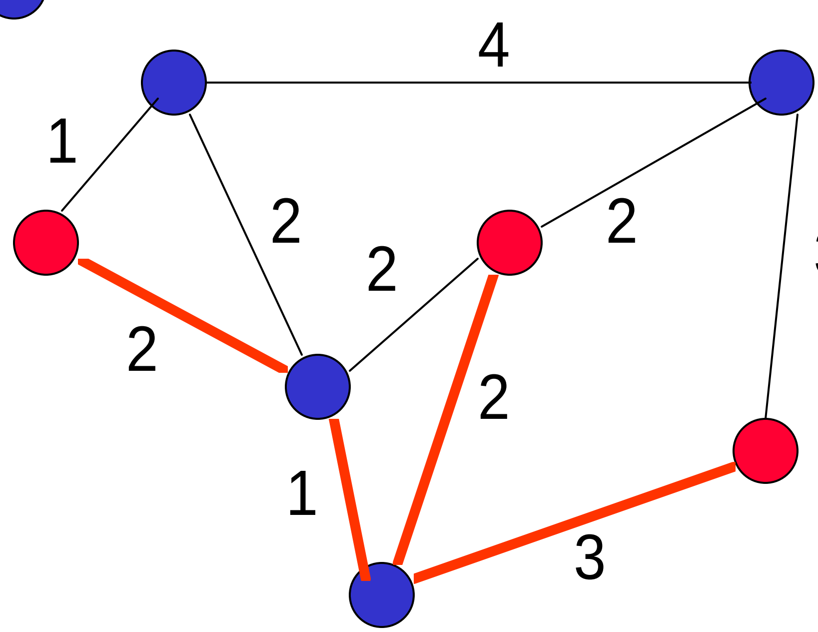
- Classic combinatorial optimization problem (Hwang, Richards, & Winter, 1992)
- An example:



Steiner Problem in Graphs



total weight = 9



total weight = 8
(optimum)

Steiner Problem in Graphs: Complexity

- NP-complete (Karp, 1972)
- Remains NP-complete for:
  - grid graphs
  - bipartite graphs
  - chordal & 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 in the books by Voss (1990), Hwang, Richards, & Winter (1992), and Du & Pardalos (1993).

GRASP

Feo & Resende (1989, 1995)

```
best_obj = 0;
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 & 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 to find local minimum

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

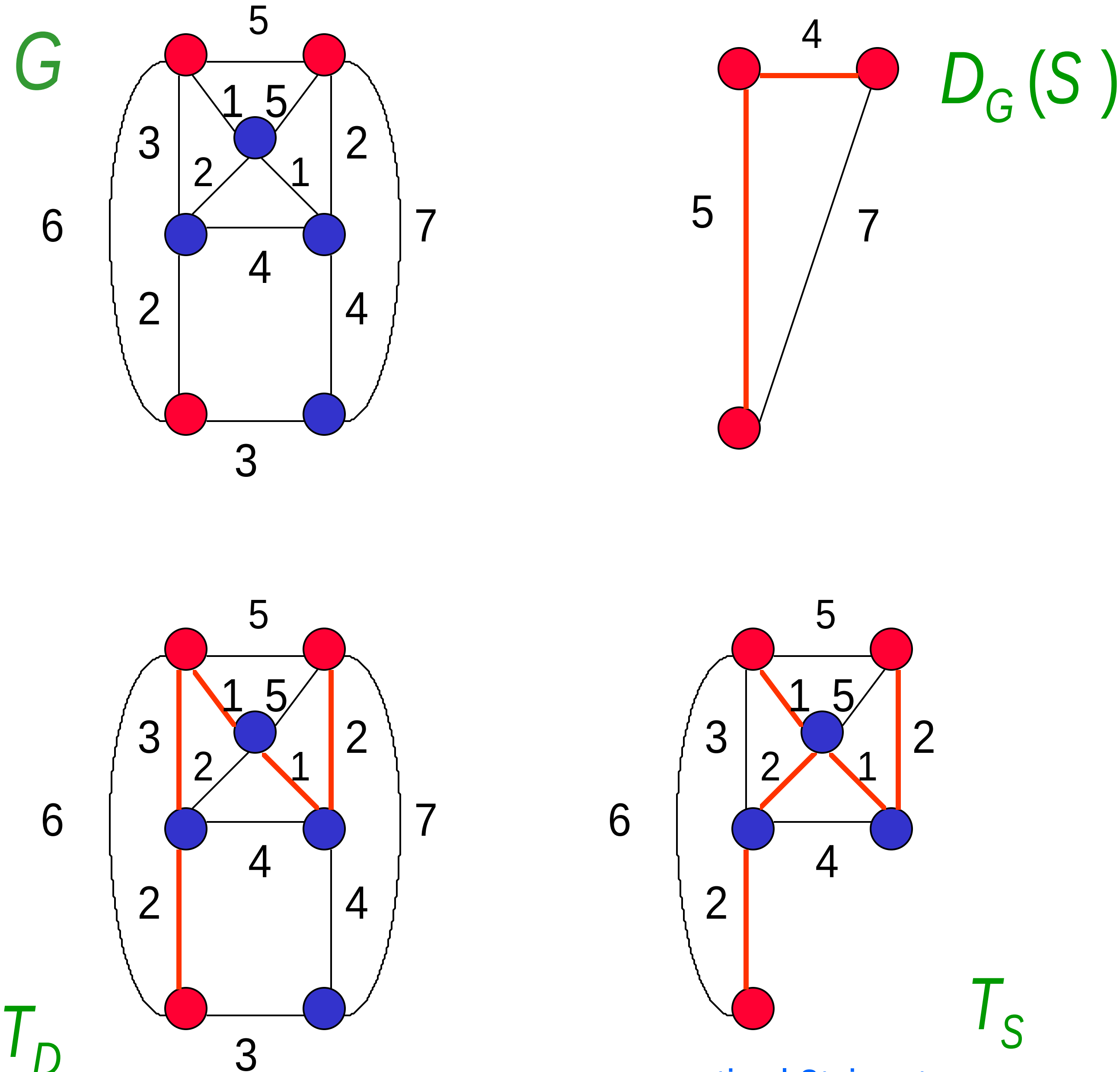
Spanning tree based construction procedure

- First, we describe the spanning tree based construction procedure
  - Based on distance network heuristic (Choukhmane, 1978; Kou et al., 1981; Plesník, 1981; Iwainsky et al., 1986)
  - Uses distance modification of Mehlhorn (1988)
  - Uses randomized variant of Kruskal’s minimum spanning tree (MST) algorithm (1956)

Spanning tree based construction procedure

- Distance network heuristic
  1) Construct distance network $D_G(S)$
  2) Compute a MST of $D_G(S)$
  3) Construct graph T_D, from the MST by replacing each edge of the MST by a shortest path in G (obs: T_D can always be a tree)
  4) Consider G_T, the subgraph of G induced by the vertices of T_D. Compute a MST of G_T : T_S .
  5) Delete from T_S the non-terminals of degree 1, one at a time.

Distance network heuristic (DNH)



optimal Steiner tree

Mehlhorn's modification

- Mehlhorn adapted the DNH by replacing the metric used to build the distance matrix, improving its complexity.
- For all $s \in S$, $N(s)$ contains the non-terminal vertices of V that are closer to s than to any other vertex in S .
- The graph $G'(S,E')$ is defined, where
  - $E' = \{ (s,t) \mid s,t \in S$ and $\exists\ (u,v) \in V$ such that $u \in N(s), v \in N(t) \}$
  - $w'(s,t) = \min \{d(s,u) + w(u,v) + d(v,t)\}$
- MTS (G') is also a MST $(D_G(S))$

Making DNH into a GRASP construction method

- In Kruskal's algorithm, instead of selecting the least weight feasible edge:
  - Build a restricted candidate list (RCL) consisting of low weight edges.
  - Select, at random, an edge from the RCL.

$$\text{RCL} = \{\, e \ni w(e) < \underline{w} + a\,(\overline{w} - \underline{w}) \,\}$$

smallest weight ($\underline{w}$)

largest weight ($\overline{w}$)

$$0 \leq a \leq 1$$

Local Search Procedures

- We consider two local search methods:
  - vertex based approach
  - path based approach

Vertex based local search

- The neighbors of T_S are all MST obtained
  - by adding to T_S a non-terminal vertex not in T_S
  - by deleting from T_S a non-terminal vertex that is in T_S
- Weights used in MST computations are of the original graph.

Vertex based local search

- Given a MST T of graph G, the computation of a new MST of the graph $G + \{v\}$ can be done in $O(|V|)$ time (Minoux, 1990)
- To compute a MST of $G - \{v\}$, we use Kruskal's algorithm.

Vertex based local search

```
LocalSearch (T )
{
    for all v  not in T {
            compute cost of MST T'  with v  inserted;
    }
    if cost(best T' ) < cost(T ) {
            T = best T' ;
            LocalSearch(T );
    }
```

Vertex based local search

```
else{ /* if no improvement in insertion is possible */
     for all v  in T {
            compute cost of MST T'  with v  deleted;
    }
     if cost(best T' ) < cost(T ) {
            T = best T' ;
            LocalSearch(T );
    }
}
```

Path based local search

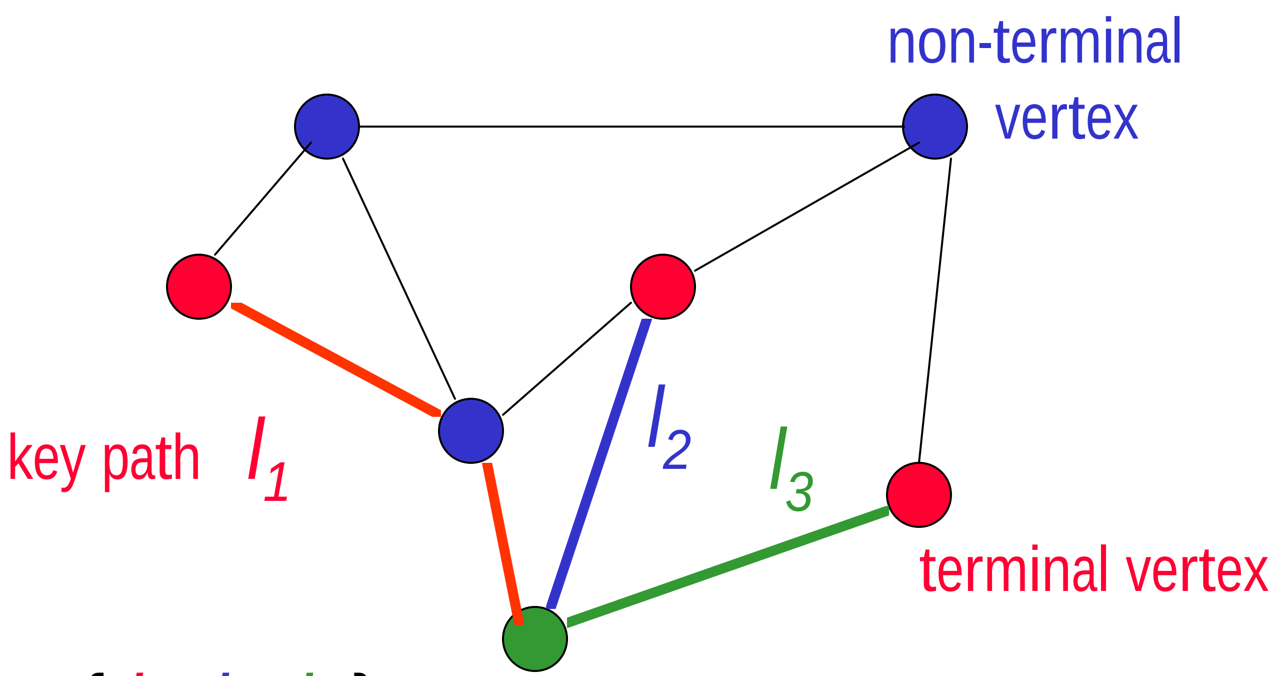
- We use the key path exchange local search of Verhoeven et al. (1996)

Path based local search

- We need two definitions
  - A key vertex is a Steiner vertex with degree at least 3
  - A key path is a path in a Steiner tree T_S of which all intermediate vertices are Steiner vertices with degree 2 in T_S, and whose end vertices are either terminal or key vertices.
- A Steiner tree has at most
  - $|S| - 2$ key vertices
  - $2|S| - 3$ key paths

Path based construction procedure

- A minimal Steiner tree consists of key paths that are shortest paths between key vertices or terminals.



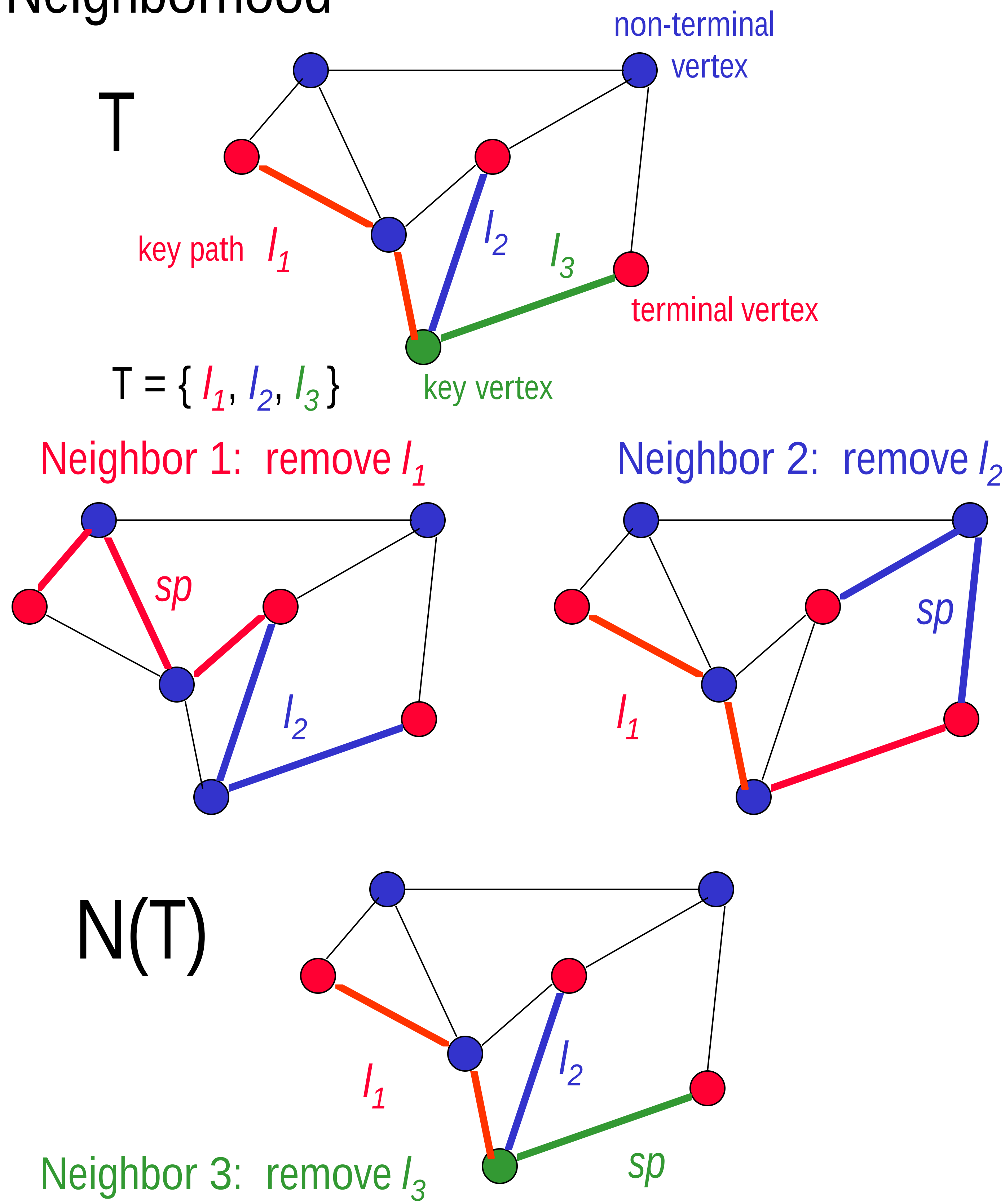
$T = \{ l_1, l_2, l_3 \}$

Path based local search

- Let
  - $T = \{l_1, l_2, \ldots, l_K\}$ be a Steiner tree.
  - C_i and C'_i be the 2 components that result from the removal of l_i from T.
- The neighborhood $N(T)$ =

  $\{C_i \cup C'_i \cup sp\,(C_i, C'_i) \mid i = 1,2,\ldots,K\}$
    - Observe that $C_i \cup C'_i \cup l_i = T$
- $N(T)$ contains at most $2|S|-3$ neighbors.

Neighborhood



Path based local search

```
LocalSearch (T = {l_1, l_2, ..., l_K} )
{   for (i =1,...,K ) {
        if ( sp (C_i ,C'_i ) < length ( l_i ) ){
            T = T - { l_i } U sp (C_i ,C'_i )
             if necessary{
                 update T  to be a set of key paths
             }
             LocalSearch (T )
        }
    }
}
```

Path based local search

- Solutions only have neighbors with lower or equal cost.
- A replacement of a key path in *T* can lead to the same Steiner tree if no shorter path exists.
    - This implies that local minima have no neighbors.

Comparing the two local search strategies

- Use John Beasley's OR-Library
  - OR-Library: series C & D
  - reduced graphs using reduction tests of Duin & Vogenant (1989)
- IBM RISC/6000 390
- C implementation
  - IBM xlC compiler v. 3.1.3 with flags –O3 –qstrict
- 512 GRASP iterations
- fixed RCL parameter $\alpha = 0.1$

Comparing the two local search strategies

vertex-based local search

Series	# opt	avg err.	max err.	cpu time
C	18/20	0.17%	2.65%	52.23s
D	14/20	0.26%	2.24%	225.51s

path-based local search

Series	# opt	avg err.	max err.	cpu time
C	15/20	0.39%	3.13%	27.02s
D	10/20	0.54%	4.47%	114.60s

Comparing the two local search strategies

- Both variants find good solutions
- Optimal solutions found on large portion of instances
- Average error < 1% off optimal
- Worst error < 5% off optimal
- Node-based neighborhood produces best-quality solutions
- Computation time of node-based neighborhood search is twice that of path-based neighborhood search

Hybrid local search

- repeat
  - T = GRASP construction
  - If T is a new Steiner tree (hashing)
    - T = path-based search (T)
    - if w(T) < λ best weight ($\lambda > 1$)
      - T = node-based search (T)
  - if w(T) < best weight
    - save tree: $T^* = T$
    - best weight = w(T)

Simple parallelization

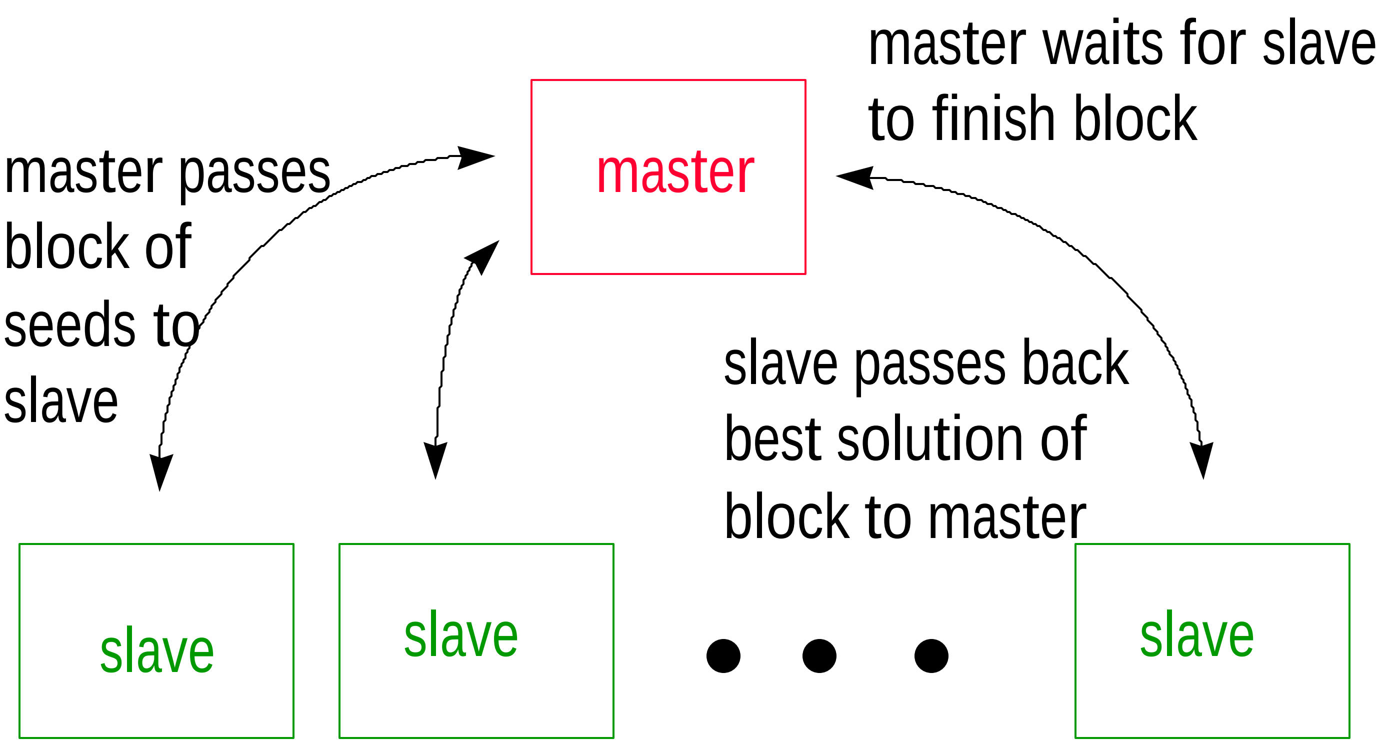
- Most straightforward scheme for parallel GRASP is distribution of iterations to different processors.
- Care is required so that two iterations never start off with same random number generator seed.
  - run generator and record all N_g seeds in seed( ) array
  - start iteration *i* with seed seed(*i*)

MPI implementation

IBM SP 2 with 32 RS6000 390 processors
with 256 Mbytes of memory each

We used *p* = 1,2,4,8,16 slave processors, each running a total of 512/*p* GRASP iterations with $\lambda = 1.01$

OR-library test problems: Series C, D, & E



Computational results: hybrid local search

Series C: 500 nodes, 625 to 12500 edges, and 5 to 250 terminal nodes

Series D: 1000 nodes, 1250 to 25000 edges, and 5 to 500 terminal nodes

Series E: 2500 nodes, 3125 to 62500 edges, and 5 to 1200 terminal nodes.

Series	# opt	avg. err.	max err.
C	17/20	0.23%	3.03%
D	16/20	0.18%	2.19%
E	13/20	0.26%	3.09%

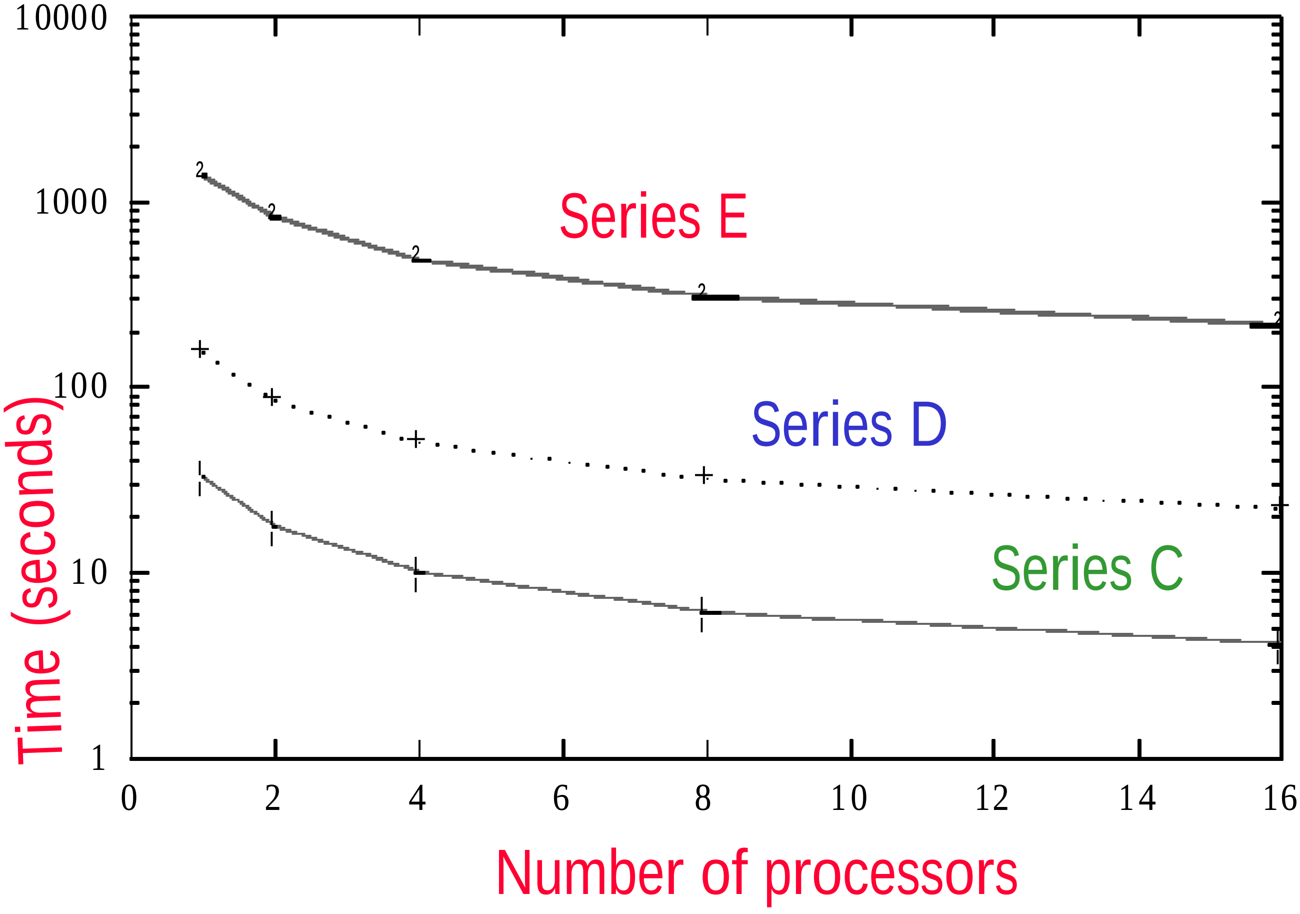
Computational results: hybrid local search

- Series C:
  - all three sub-optimal solutions found were off only by one from optimal
- Series D:
  - two of the four sub-optimal solutions found were off only by one from optimal
- Series E:
  - Of the seven sub-optimal solutions found, six were less than 1% from optimal

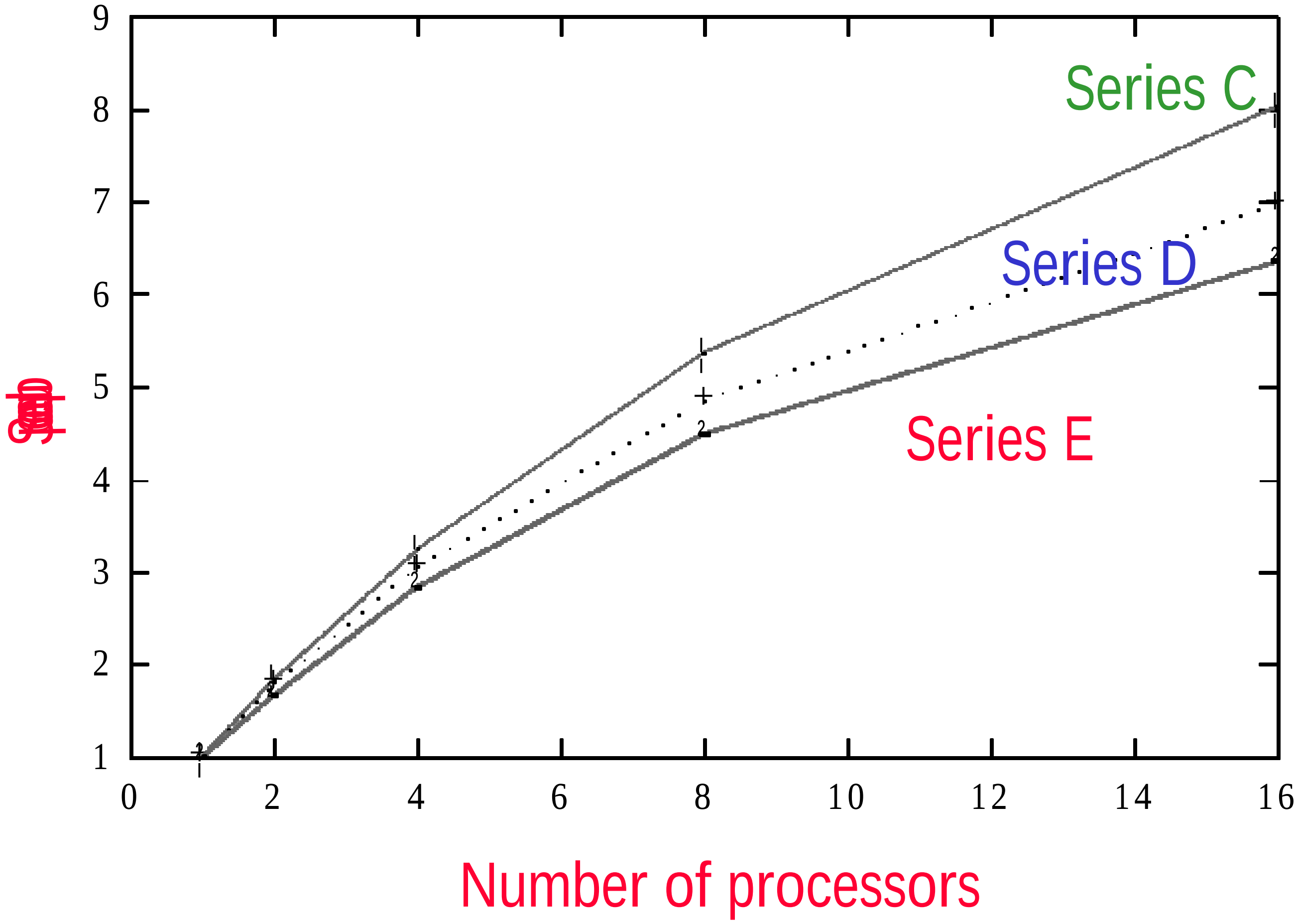
State of art

- Parallel GRASP found 46 of 60 otimal solutions
- Two state of the art Tabu Search implementations found:
  - 42 (Ribeiro & Souza, to appear in Networks)
  - 44 (Bastos & Ribeiro, MIC99)

Elapsed time



Speedup



Series C
Series D
Series E
9
8
7
6
5
4
3
2
1
0
2
4
6
8
10
12
14
16
Number of processors

Computational results: hybrid local search

- For some instances, speed up was almost linear
- For some others, parallelization did not contribute much (because of memory structures used that made sequential algorithm very fast)
- Main contribution of parallel implementation was on notably difficult instances
- With more than 16 processors, it appears more speedup is possible