Talk given at:

9th INFORMS Computing Society Conference

Annapolis, Maryland
January 5 to 7, 2005

A Hybrid Multistart Heuristic for the Uncapacitated Facility Location Problem

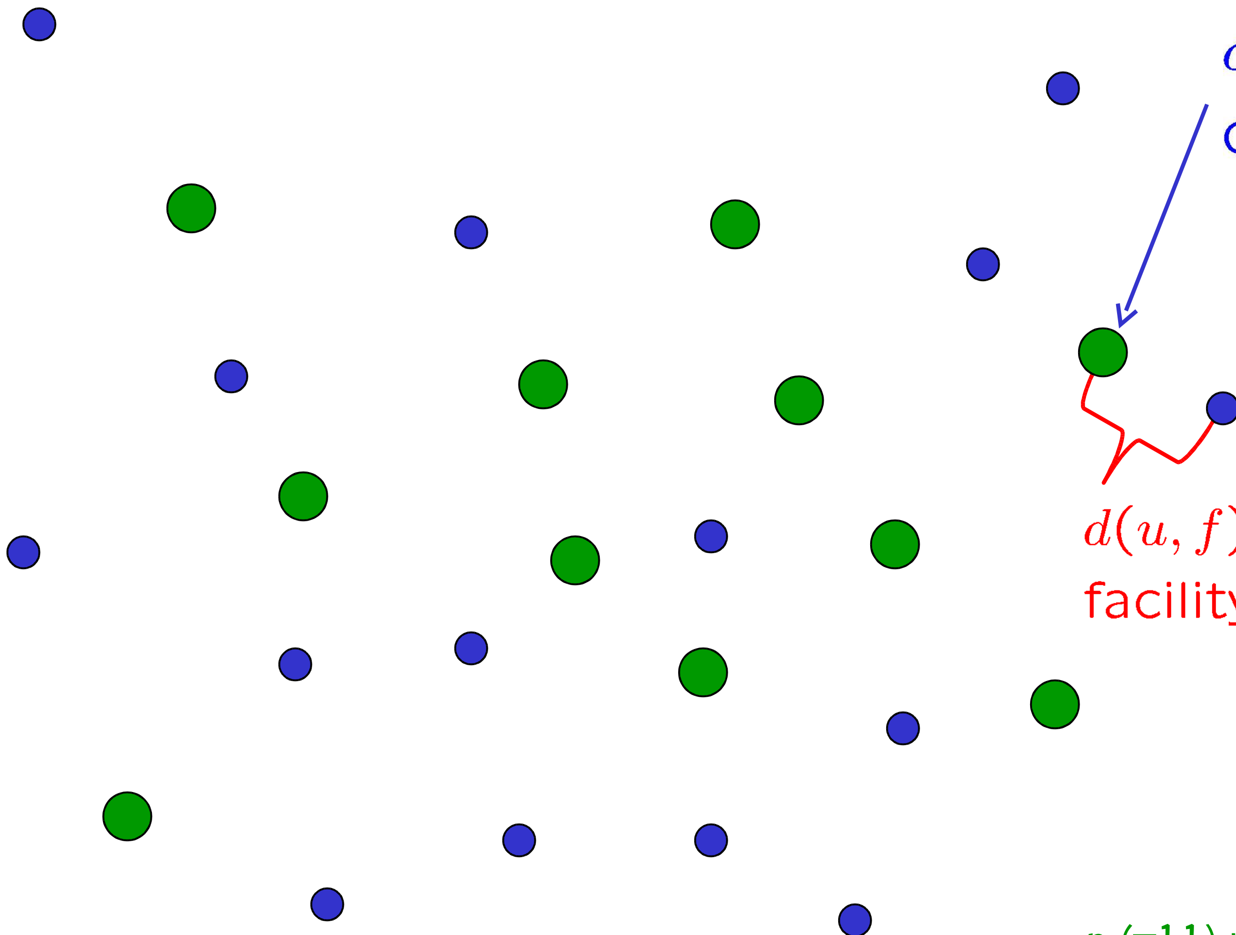
Maurício G.C. RESENDE

Alg. & Opt. Research Department
AT&T Labs Research

Renato F. WERNECK

Department of Computer Science
Princeton University

Uncapacitated facility location problem



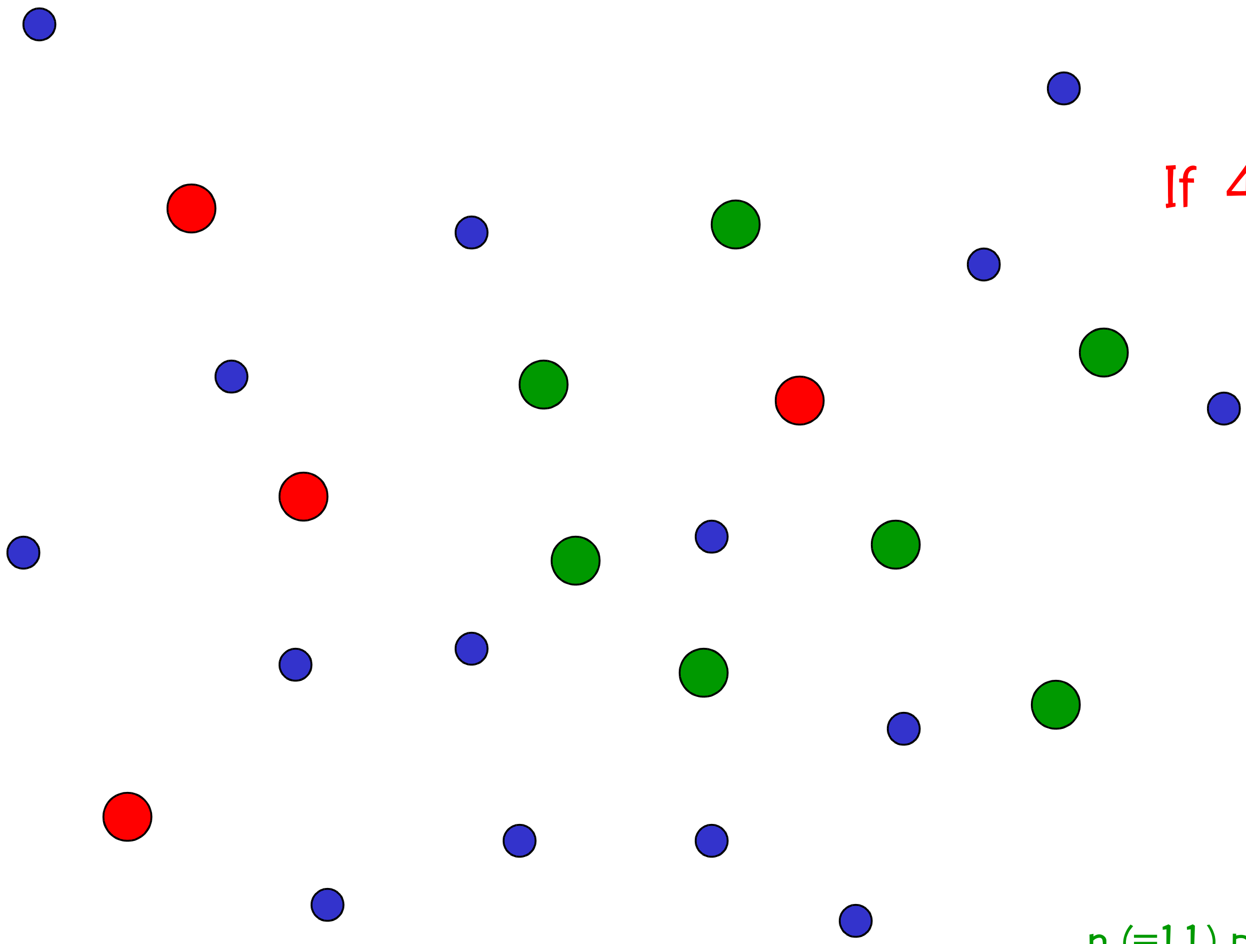
$c(f)$: setup cost of opening facility f

$d(u, f)$: distance between facility d and user u.

n (=11) potential facility locations

m (=15) customers

Uncapacitated facility location problem

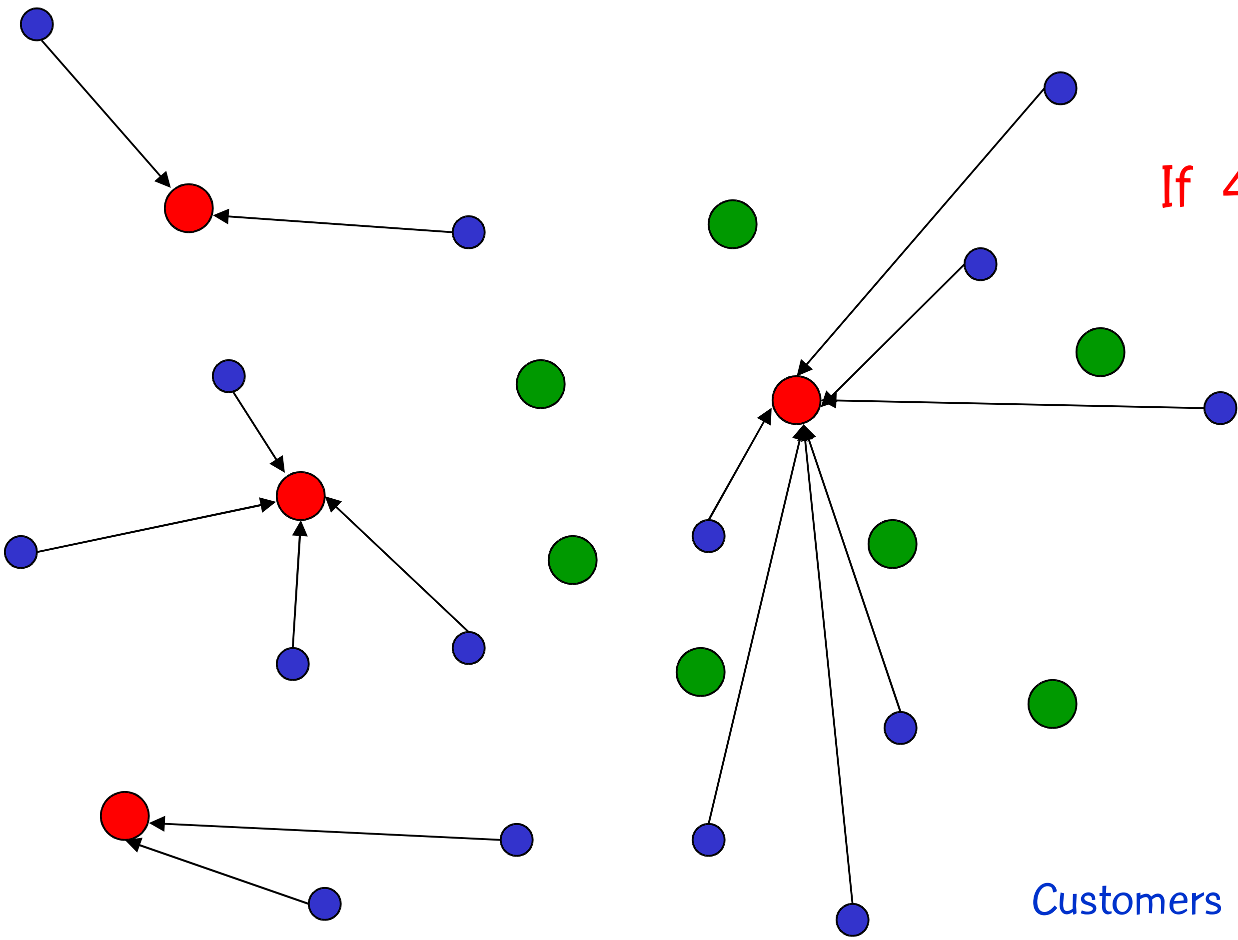


If 4 facilities are opened

n (=11) potential facility locations

m (=15) customers

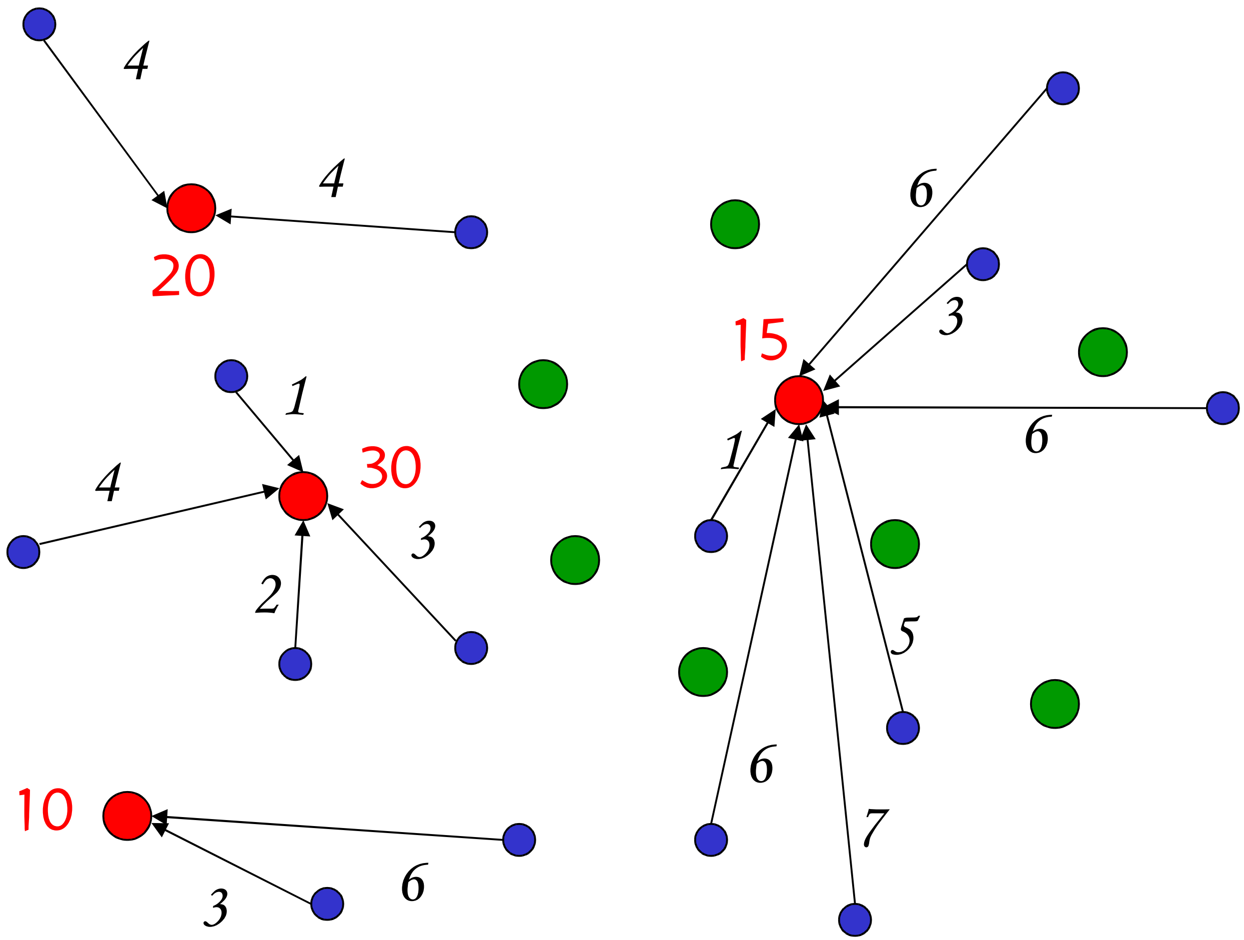
Uncapacitated facility location problem



If 4 facilities are opened

Customers home into nearest open facility.

Uncapacitated facility location problem

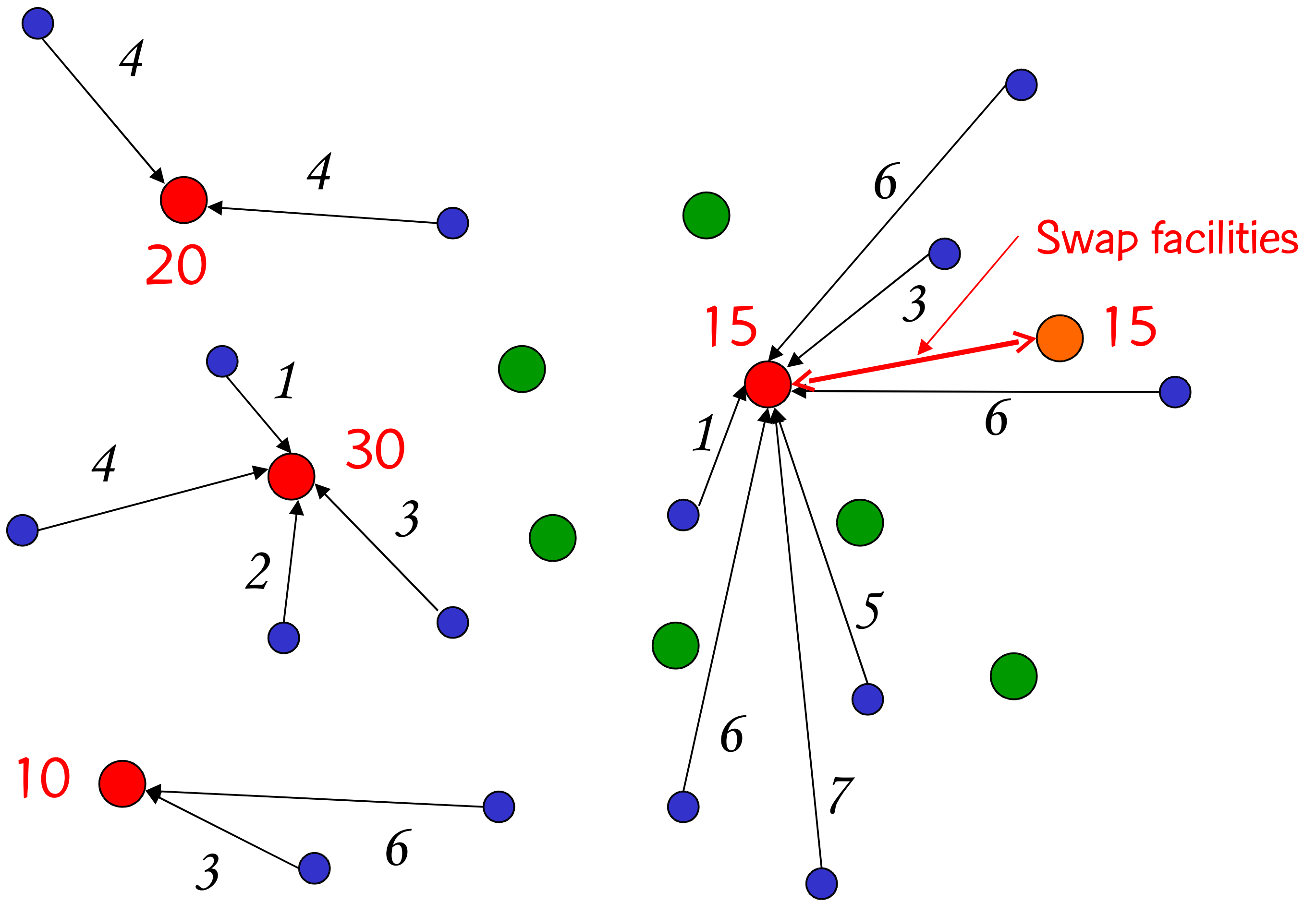


Objective of optimization:

Minimize sum of the distances between customers and their nearest open facility plus the cost of opening the facilities.

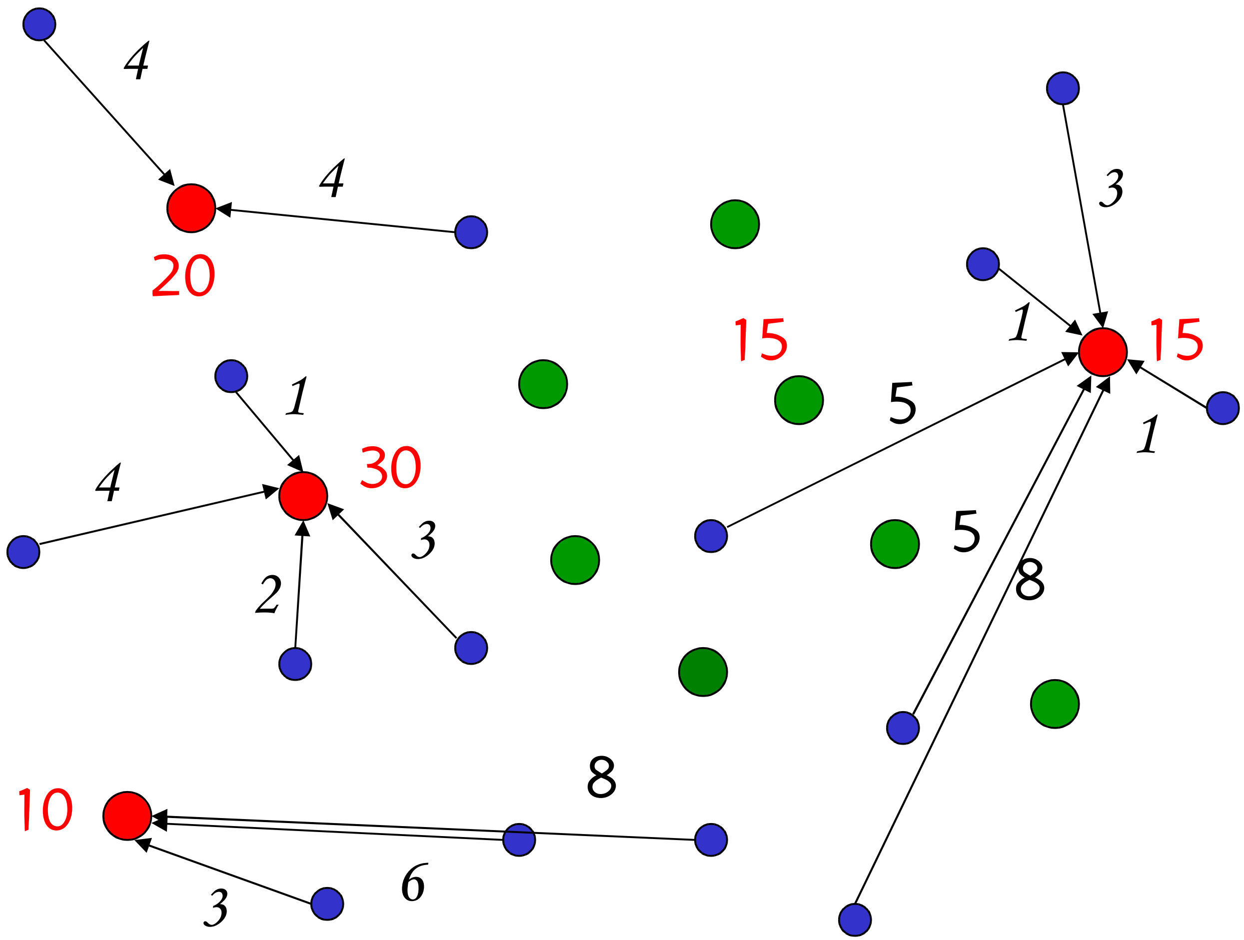
Total cost = 61+ 75 = 136

Uncapacitated facility location problem



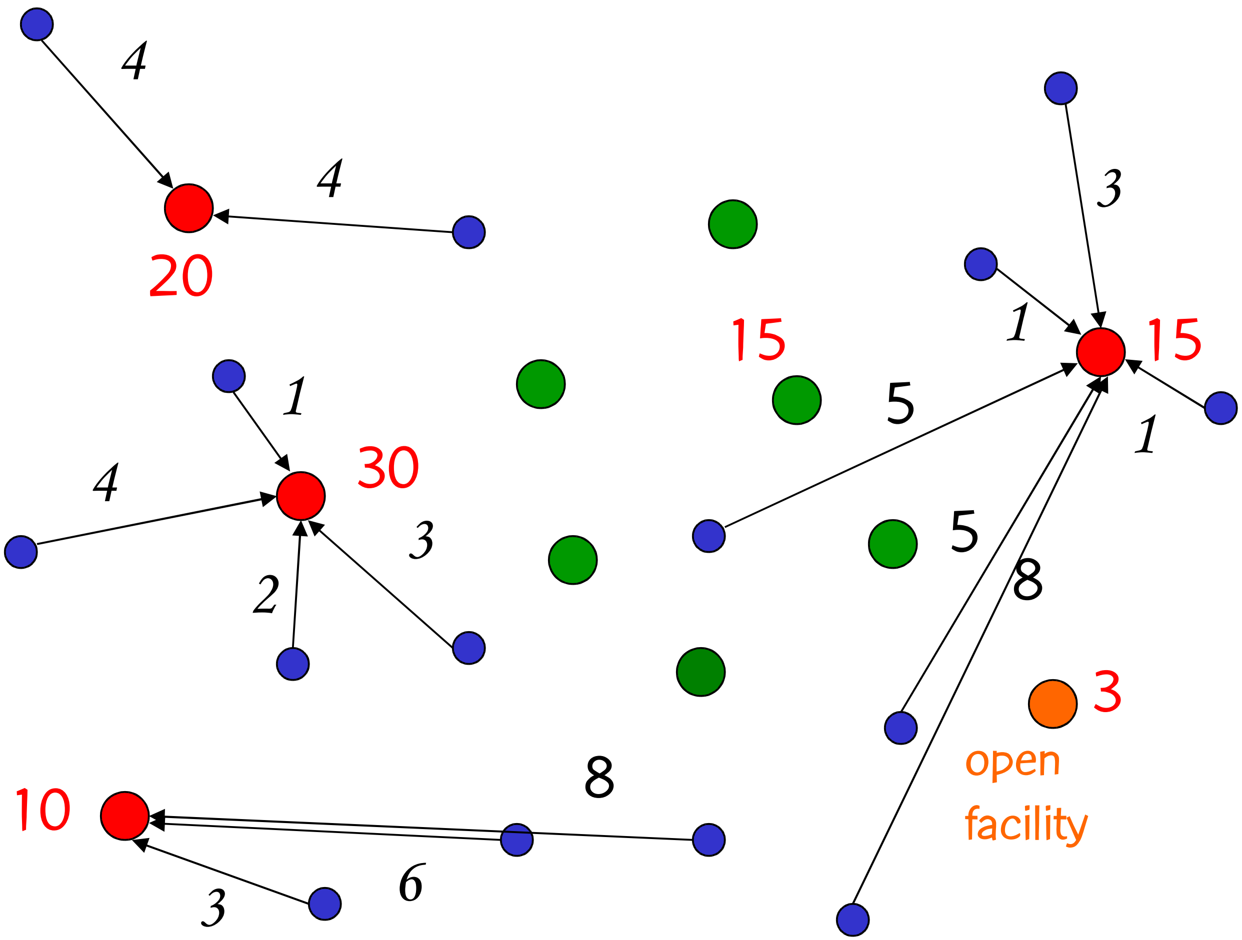
Total cost = 61+ 75 = 136

Uncapacitated facility location problem



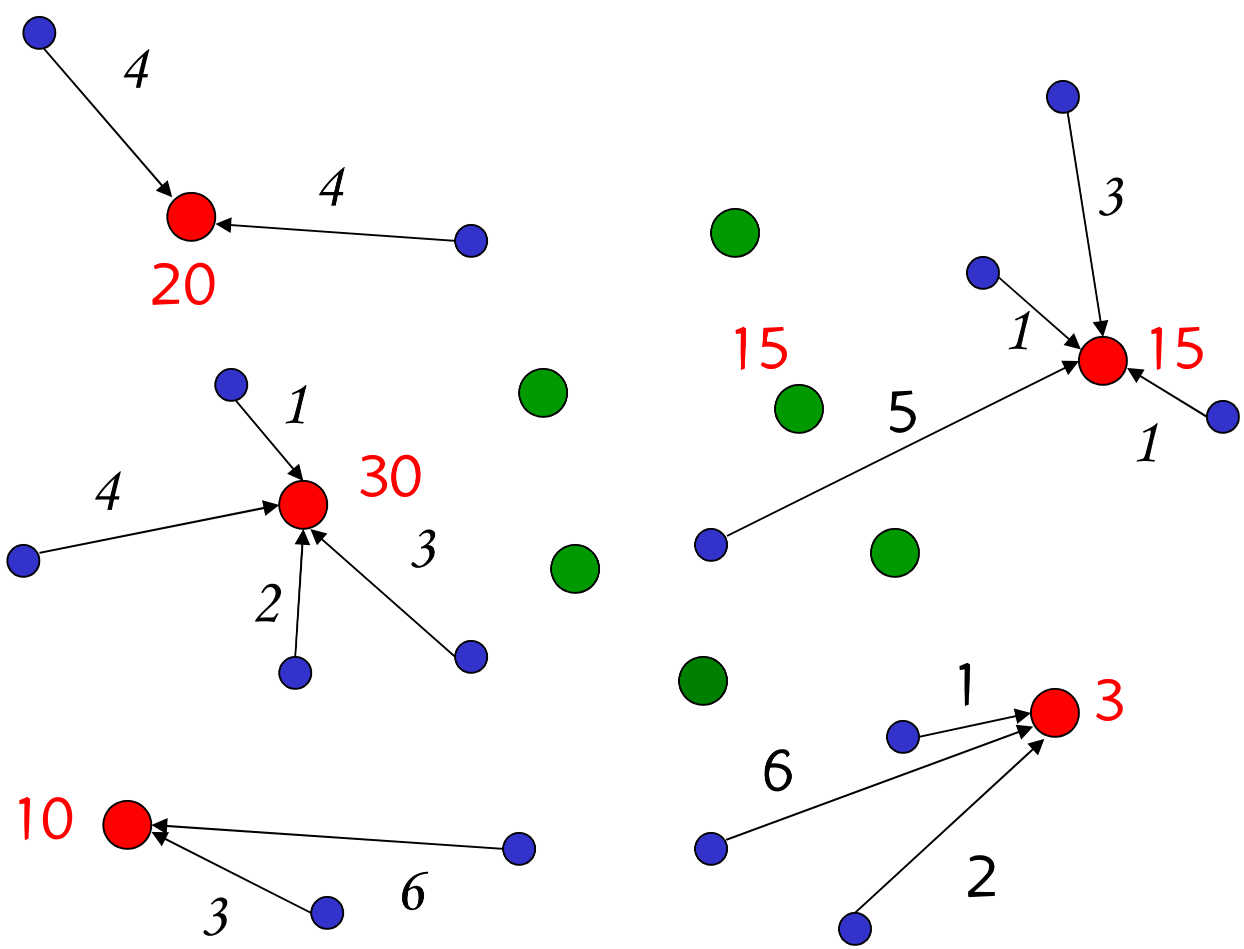
Total cost = 58 + 75 =
133 < 136

Uncapacitated facility location problem



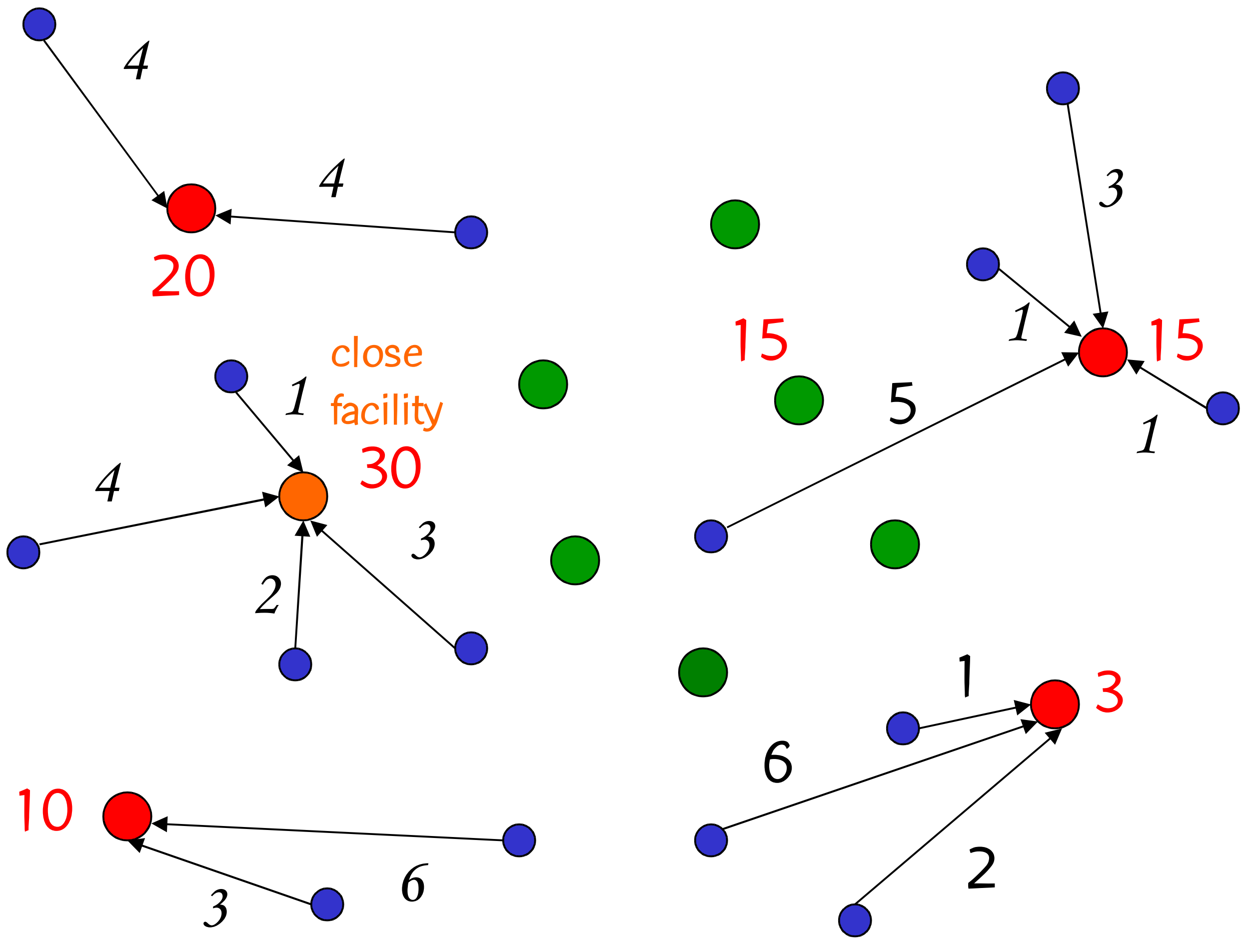
Total cost = 58 + 75 = 133 < 136

Uncapacitated facility location problem



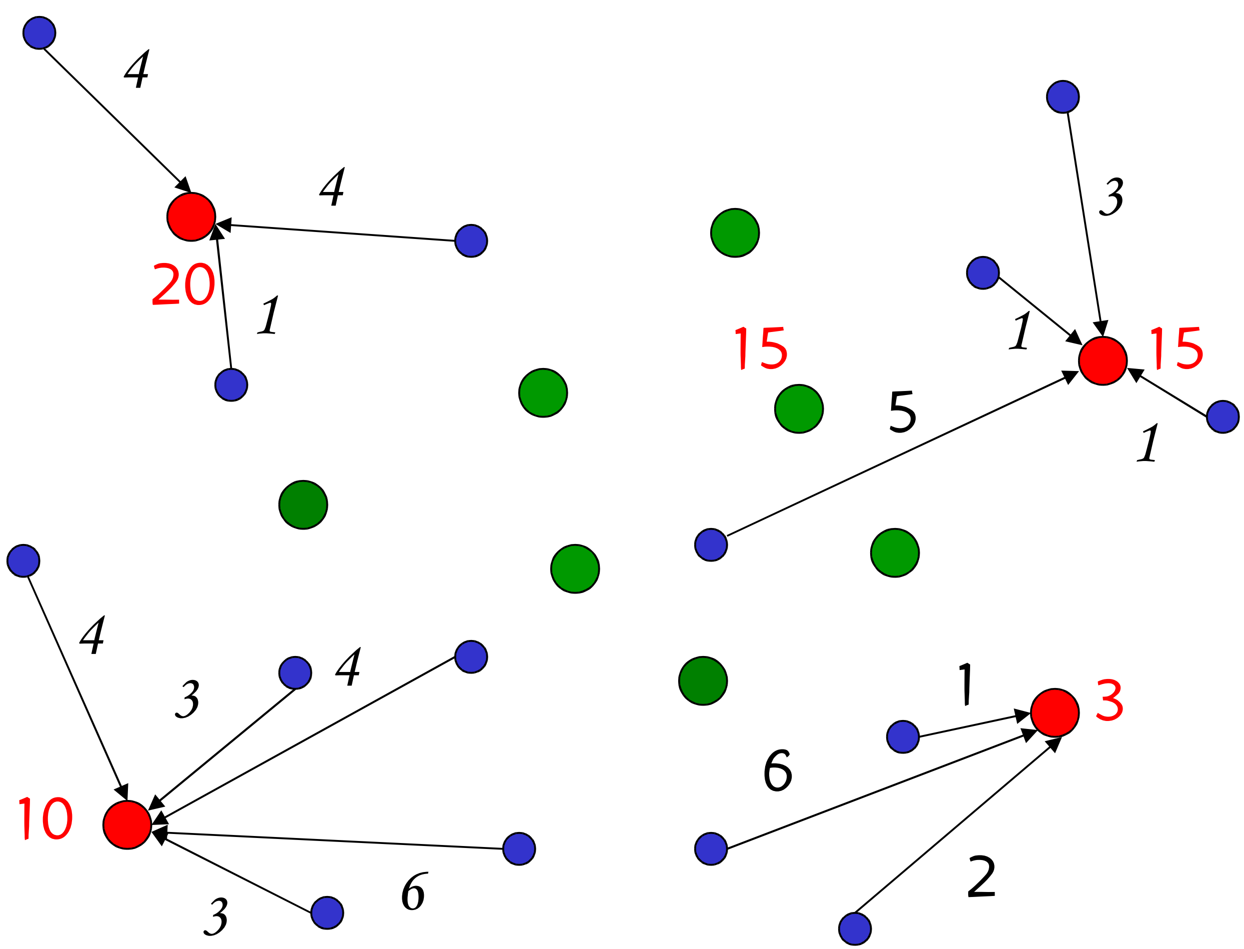
Total cost = 46 + 78 = 124 < 133

Uncapacitated facility location problem



Total cost = 46 + 78 = 124 < 133

Uncapacitated facility location problem



Total cost = 48 + 48 = 96 < 124

Facility location problem

- Set F of potential facilities, each facility f with a setup cost c(f)
- Set U of users that must be served by a facility. The cost of serving user u by facility f is d(u,f)
  - Facility location problem: Determine a set of facilities $S \subseteq F$ to open so as to minimize the total cost:
    - $\text{cost}(S) = \sum(c(f) : f \in S) + \sum[\min\{d(u,f) : f \in S\} : u \in U]$

Facility location problem

- Set F of potential facilities, each facility f with a setup cost c(f)
- Set U of users that must be served by a facility. The cost of serving user u by facility f is d(u,f)
  - Facility location problem: Determine a set of facilities $S \subseteq F$ to open so as to minimize the total cost:
    - $\mathrm{cost}(S) = \sum(c(f) : f \in S) + \sum[\min\{d(u,f) : f \in S\} : u \in U]$

Facility location problem

- Set F of potential facilities, each facility f with a setup cost c(f)
- Set U of users that must be served by a facility. The cost of serving user u by facility f is d(u,f)
- Facility location problem: Determine a set of facilities $S \subseteq F$ to open so as to minimize the total cost:
  - $\text{cost}(S) = \sum(c(f) : f \in S) + \sum[\min\{d(u,f) : f \in S\} : u \in U]$

Uncapacitated facility location

- Customers home in to nearest open facility
- No limit on number of open facilities
- NP hard [Cournéjols, Nemhauser, & Wolsey, 1990]
- Perhaps the most common location problem, studied widely in literature both in theory & practice

Uncapacitated facility location

- Customers home in to nearest open facility
- No limit on number of open facilities
- NP hard [Cournéjols, Nemhauser, & Wolsey, 1990]
- Perhaps the most common location problem, studied widely in literature both in theory & practice

Uncapacitated facility location

- Customers home in to nearest open facility
- No limit on number of open facilities
- NP hard [Cournéjols, Nemhauser, & Wolsey, 1990]
- Perhaps the most common location problem, studied widely in literature both in theory & practice

Uncapacitated facility location

- Customers home in to nearest open facility
- No limit on number of open facilities
- NP hard [Cournéjols, Nemhauser, & Wolsey, 1990]
- Perhaps the most common location problem, studied widely in literature both in theory & practice

Uncapacitated facility location

- Exact methods exist, e.g. [Conn and Cournéjols, 1990; Körkel, 1989]
- NP-hard nature makes heuristics a natural choice for larger instances
- Shmoys, Tardos, & Aardal (1997) present a 3.16-opt approximation algorithm
- Improvements, e.g. [Jain et al., 2002, 2003; Mahdian, Ye, & Zhang, 2002] have led to polynomial-time algorithms that find a solution within a factor of around 1.5 from the optimal.

Uncapacitated facility location

- Exact methods exist, e.g. [Conn and Cournéjols, 1990; Körkel, 1989]
- NP-hard nature makes heuristics a natural choice for larger instances
- Shmoys, Tardos, & Aardal (1997) present a 3.16-opt approximation algorithm
- Improvements, e.g. [Jain et al., 2002, 2003; Mahdian, Ye, & Zhang, 2002] have led to polynomial-time algorithms that find a solution within a factor of around 1.5 from the optimal.

Uncapacitated facility location

- Exact methods exist, e.g. [Conn and Cournéjols, 1990; Körkel, 1989]
- NP-hard nature makes heuristics a natural choice for larger instances
- Shmoys, Tardos, & Aardal (1997) present a 3.16-opt approximation algorithm
- Improvements, e.g. [Jain et al., 2002, 2003; Mahdian, Ye, & Zhang, 2002] have led to polynomial-time algorithms that find a solution within a factor of around 1.5 from the optimal.

Uncapacitated facility location

- Exact methods exist, e.g. [Conn and Cournéjols, 1990; Körkel, 1989]
- NP-hard nature makes heuristics a natural choice for larger instances
- Shmoys, Tardos, & Aardal (1997) present a 3.16-opt approximation algorithm
- Improvements, e.g. [Jain et al., 2002, 2003; Mahdian, Ye, & Zhang, 2002] have led to polynomial-time algorithms that find a solution within a factor of around 1.5 from the optimal.

Uncapacitated facility location

- Unfortunately, there is not much more room for improvement: Guha & Khuller (1999) established a lower bound of 1.463 for the approximation factor.
- In practice, approximation algorithms tend to be much closer for non-pathological instances: The 1.61-opt algorithm of Jain et al. (2003) was always within 2% of optimal in their experiments.
- Though interesting in theory, approximation algorithms are often outperformed in practice by more straightforward heuristics with no particular performance guarantees.

Uncapacitated facility location

- Unfortunately, there is not much more room for improvement: Guha & Khuller (1999) established a lower bound of 1.463 for the approximation factor.
- In practice, approximation algorithms tend to be much closer for non-pathological instances: The 1.61-opt algorithm of Jain et al. (2003) was always within 2% of optimal in their experiments.
- Though interesting in theory, approximation algorithms are often outperformed in practice by more straightforward heuristics with no particular performance guarantees.

Uncapacitated facility location

- Unfortunately, there is not much more room for improvement: Guha & Khuller (1999) established a lower bound of 1.463 for the approximation factor.
- In practice, approximation algorithms tend to be much closer for non-pathological instances: The 1.61-opt algorithm of Jain et al. (2003) was always within 2% of optimal in their experiments.
- Though interesting in theory, approximation algorithms are often outperformed in practice by more straightforward heuristics with no particular performance guarantees.

Uncapacitated facility location

- Pioneering work on heuristics: Kuehn & Hamburger (1963)
- Since then, more sophisticated heuristics have been applied:
  - Simulated annealing [Alves & Almeida, 1992]
  - Genetic algorithms [Kratica et al., 2001]
  - Tabu search [Ghosh, 2003; Michel & Van Hentenryck, 2003]
  - Complete local search with memory [Ghosh, 2003]
- Dual-based methods have also shown promising results:
  - Dual ascent [Erlenkotter, 1978]
  - Lagrangean dual ascent [Guignard, 1988]
  - Volume algorithm [Barahona & Chudak, 1999]

Uncapacitated facility location

- Pioneering work on heuristics: Kuehn & Hamburger (1963)
- Since then, more sophisticated heuristics have been applied:
  - Simulated annealing [Alves & Almeida, 1992]
  - Genetic algorithms [Kratica et al., 2001]
  - Tabu search [Ghosh, 2003; Michel & Van Hentenryck, 2003]
  - Complete local search with memory [Ghosh, 2003]
- Dual-based methods have also shown promising results:
  - Dual ascent [Erlenkotter, 1978]
  - Lagrangean dual ascent [Guignard, 1988]
  - Volume algorithm [Barahona & Chudak, 1999]

Uncapacitated facility location

- Pioneering work on heuristics: Kuehn & Hamburger (1963)
- Since then, more sophisticated heuristics have been applied:
  - Simulated annealing [Alves & Almeida, 1992]
  - Genetic algorithms [Kratica et al., 2001]
  - Tabu search [Ghosh, 2003; Michel & Van Hentenryck, 2003]
  - Complete local search with memory [Ghosh, 2003]
- Dual-based methods have also shown promising results:
  - Dual ascent [Erlenkotter, 1978]
  - Lagrangean dual ascent [Guignard, 1988]
  - Volume algorithm [Barahona & Chudak, 1999]

Uncapacitated facility location

- Hofer (2002) presented computational comparison of five methods:
  - JMS, an approximation algorithm of Jain et al. (2002)
  - MYZ, an approximation algorithm of Mahdian et al. (2002)
  - A swap-based local search
  - Tabu search of Michel & Van Hentenryck (2003)
  - Volume algorithm of Barahona & Chudack (1999)
- Hofer's conclusion: tabu search finds best solutions in reasonable time and is recommended to practitioners.

Uncapacitated facility location

- Hofer (2002) presented computational comparison of five methods:
  - JMS, an approximation algorithm of Jain et al. (2002)
  - MYZ, an approximation algorithm of Mahdian et al. (2002)
  - A swap-based local search
  - Tabu search of Michel & Van Hentenryck (2003)
  - Volume algorithm of Barahona & Chudack (1999)
- Hofer's conclusion: tabu search finds best solutions in reasonable time and is recommended to practitioners.

Our algorithm

- In this talk, we provide an alternative that can be even better in practice.
- It is a hybrid multistart heuristic akin to the one we developed in Resende & Werneck (2004) for the p-median problem
- A series of minor adaptations is enough to build a very robust algorithm, capable of obtaining near-optimal solutions for a wide variety of instances of the facility location problem.

Our algorithm

- In this talk, we provide an alternative that can be even better in practice.
- It is a hybrid multistart heuristic akin to the one we developed in Resende & Werneck (2004) for the p-median problem
- A series of minor adaptations is enough to build a very robust algorithm, capable of obtaining near-optimal solutions for a wide variety of instances of the facility location problem.

Our algorithm

- In this talk, we provide an alternative that can be even better in practice.
- It is a hybrid multistart heuristic akin to the one we developed in Resende & Werneck (2004) for the p-median problem
- A series of minor adaptations is enough to build a very robust algorithm, capable of obtaining near-optimal solutions for a wide variety of instances of the facility location problem.

Our algorithm

- Works in two phases:
  - Multistart routine with intensification: Each iteration builds a randomized solution and applies local search to it. The resulting solution S is combined, in a process called path-relinking, with another solution from a set of elite solutions, resulting in S'. The algorithm tries to insert S and S' into the elite set.
  - Post-optimization: Solutions from the elite set are combined with each other in a process that hopefully results in better solutions.
- The method is called HYBRID because it combines elements of several metaheuristics.

Our algorithm

- Works in two phases:
  - Multistart routine with intensification: Each iteration builds a rondomized solution and applies local search to it. The resulting solution S is combined with a process called path-relinking with another solution from a set of elite solutions, resulting in S'. The algorithm tries to insert S and S' into the elite set.
  - Post-optimization: Solutions from the elite set are combined with each other in a process that hopefully results in better solutions.
- The method is called HYBRID because it combines elements of several metaheuristics.

Our algorithm

- Works in two phases:
  - Multistart routine with intensification: Each iteration builds a rondomized solution and applies local search to it. The resulting solution S is combined with a process called path-relinking with another solution from a set of elite solutions, resulting in S'. The algorithm tries to insert S and S' into the elite set.
  - Post-optimization: Solutions from the elite set are combined with each other in a process that hopefully results in better solutions.
- The method is called HYBRID because it combines elements of several metaheuristics.

HYBRID heuristic for location problems

```
function HYBRID (seed, maxit, elitesize)
    randomize(seed);
    init(elite, elitesize);
    for i = 1 to maxit do
        S ← randomizedBuild();
        S ← localSearch(S);
        S′ ← select(elite, S);
        if (S′ ≠ NULL) then
            S′ ← pathRelinking(S, S′);
            add(elite, S′);
        endif
        add(elite, S);
    endfor
    S ← postOptimize(elite);
    return S;
end HYBRID
```

Reuse of p-median heuristic

- Although the HYBRID heuristic was originally proposed for the p-median problem, its framework can be applied to other problems: in this case, facility location.
- Recall that the p-median problem is very similar to facility location: the only difference is that instead of assigning costs to facilities, the p-median problem must specify p, the exact number of facilities to be opened.
- With minor adaptations, we can reuse several of the components used in Resende & Werneck (2004), such as the construction algorithms, local search, and path-relinking.

Reuse of p-median heuristic

- Although the HYBRID heuristic was originally proposed for the p-median problem, its framework can be applied to other problems: in this case, facility location.
- Recall that the p-median problem is very similar to facility location: the only difference is that instead of assigning costs to facilities, the p-median problem must specify p, the exact number of facilities to be opened.
- With minor adaptations, we can reuse several of the components used in Resende & Werneck (2004), such as the construction algorithms, local search, and path-relinking.

Reuse of p-median heuristic

- Although the HYBRID heuristic was originally proposed for the p-median problem, its framework can be applied to other problems: in this case, facility location.
- Recall that the p-median problem is very similar to facility location: the only difference is that instead of assigning costs to facilities, the p-median problem must specify p, the exact number of facilities to be opened.
- With minor adaptations, we can reuse several of the components used in Resende & Werneck (2004), such as the construction algorithms, local search, and path-relinking.

Paper on HYBRID for p-median

M.G.C. Resende and R.F. Werneck, A hybrid heuristic for the p-median problem, Journal of Heuristics, vol. 10, pp. 59-88, 2004.

`http://www.research.att.com/~mgcr/doc/hhpmedian.pdf`

Construction heuristic

- At iteration i, we determine the number p_i of facilities to open.
  - For $i = 1$, $p_i = \lceil m/2 \rceil$;
  - For $i > 1$, we pick the average number of facilities opened in the first $i - 1$ iterations;
- We then execute procedure *sample* of the p-median variant of HYBRID:
  - At each step, choose $\lceil \log_2 (m/p_i) \rceil$ facilities uniformly at random and select the one that reduces the total cost the most.

Construction heuristic

- At iteration i, we determine the number p_i of facilities to open.
  - For $i = 1$, $p_i = \lceil m/2 \rceil$;
  - For $i > 1$, we pick the average number of facilities opened in the first $i - 1$ iterations;
- We then execute procedure sample of the p-median variant of HYBRID:
  - At each step, choose $\lceil \log_2 (m/p_i) \rceil$ facilities uniformly at random and select the one that reduces the total cost the most.

Local search

- Local search in p-median variant: given solution S, find two facilities $f_r \in S$, $f_i \notin S$ which, if swapped, leads to a better solution.
  - This keeps number of facilities constant.
  - We also allow pure insertions and pure deletions, as well as swaps.
- All possible insertions, deletions, and swaps are considered, and the best among those is performed.
- Local search stops (at local minimum) when no improving move exists.

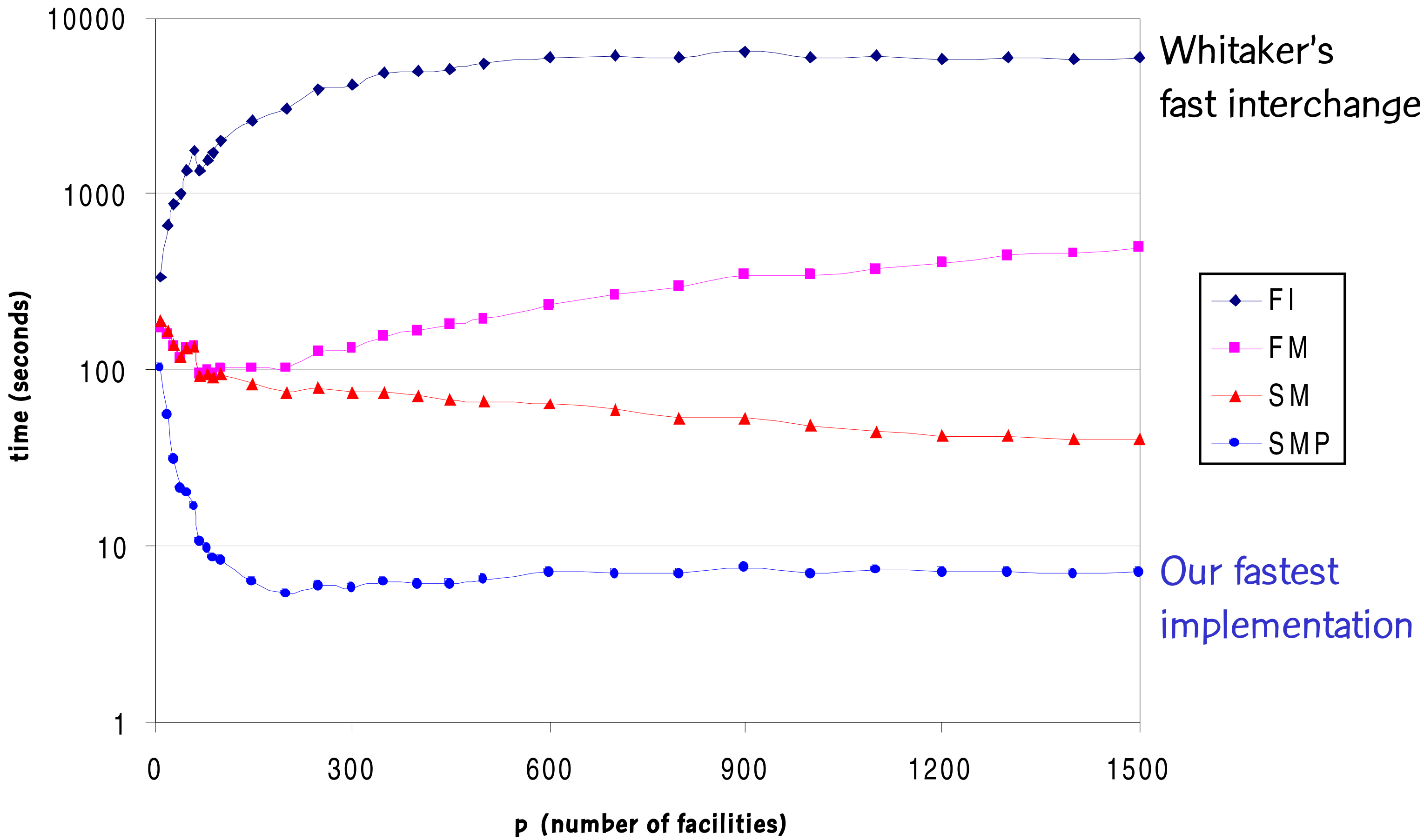
Local search

- Local search in p-median variant: given solution S, find two facilities $f_r \in S$, $f_i \notin S$ which, if swapped, leads to a better solution.
  - This keeps number of facilities constant.
  - We also allow pure insertions and pure deletions, as well as swaps.
- All possible insertions, deletions, and swaps are considered, and the best among those is performed.
- Local search stops (at local minimum) when no improving move exists.

Local search

- Local search in p-median variant: given solution S, find two facilities $f_r \in S$, $f_i \notin S$ which, if swapped, leads to a better solution.
  - This keeps number of facilities constant.
  - We also allow pure insertions and pure deletions, as well as swaps.
- All possible insertions, deletions, and swaps are considered, and the best among those is performed.
- Local search stops (at local minimum) when no improving move exists.

Local search



Largest p-median instance tested: 5934 users, Euclidean.

(preprocessing times not considered)

Local search paper

M.G.C. Resende and R.F. Werneck, A fast swap-based local search procedure for location problems, AT&T Labs Research Technical Report TD-5R3KBH, Florham Park, NJ, Sept. 2003, revised Dec. 2004. To appear in Annals of Operations Research.

http://www.research.att.com/~mgcr/doc/locationls.pdf

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

[Glover (1996)]

- Intensification: takes two solutions S_1 and S_2
- Starts from S_1 and gradually transforms it into S_2
- Operations that change solution at each step are same as in local search: insertions, deletions, swaps
- However,
  - Only facilities in $S_2 \setminus S_1$ can be inserted
  - Only facilities in $S_1 \setminus S_2$ can be removed
- At each step, most profitable move is made
- Procedure returns best local optimal in path
- If no local optimal exists, one of the extremes is returned with equal probability

Path-relinking

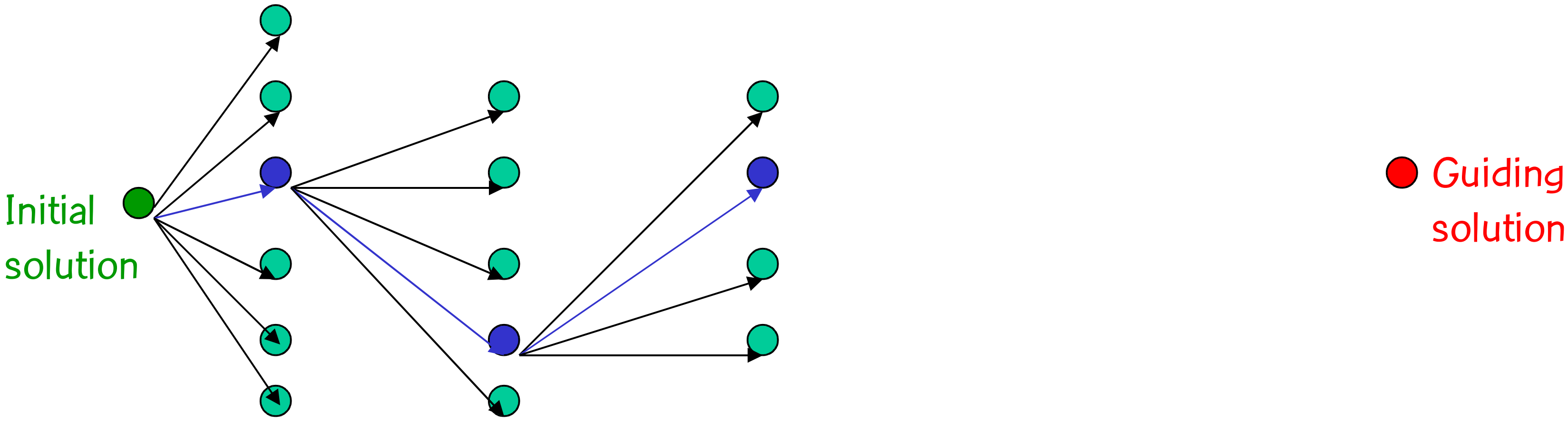
- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are analyzed and best move is taken.

Path-relinking

- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are analyzed and best move is taken.

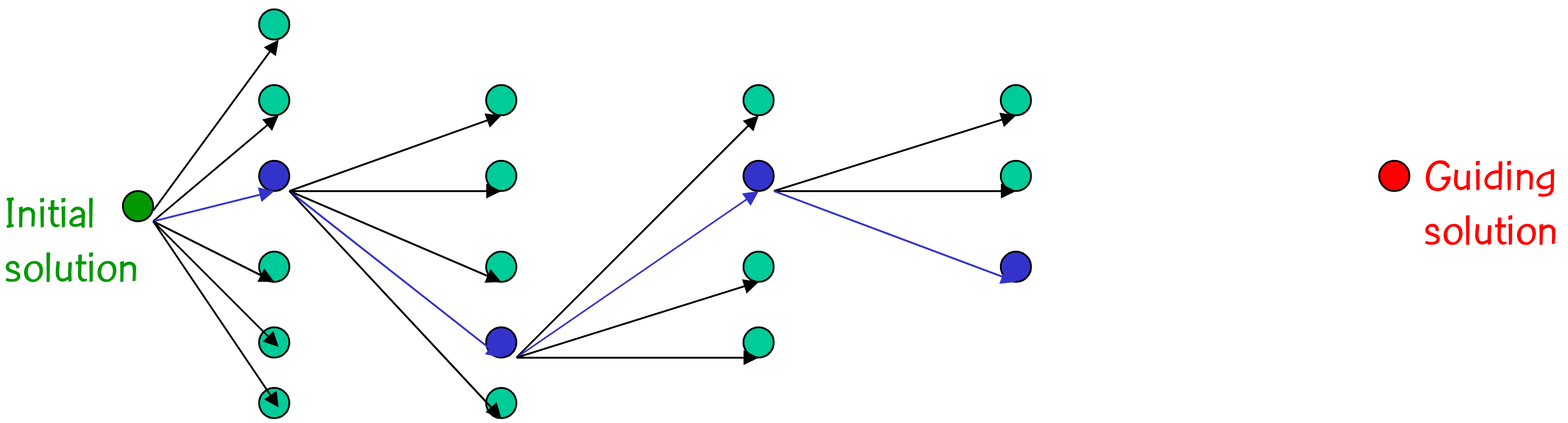
Path-relinking

- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are analyzed and best move is taken.



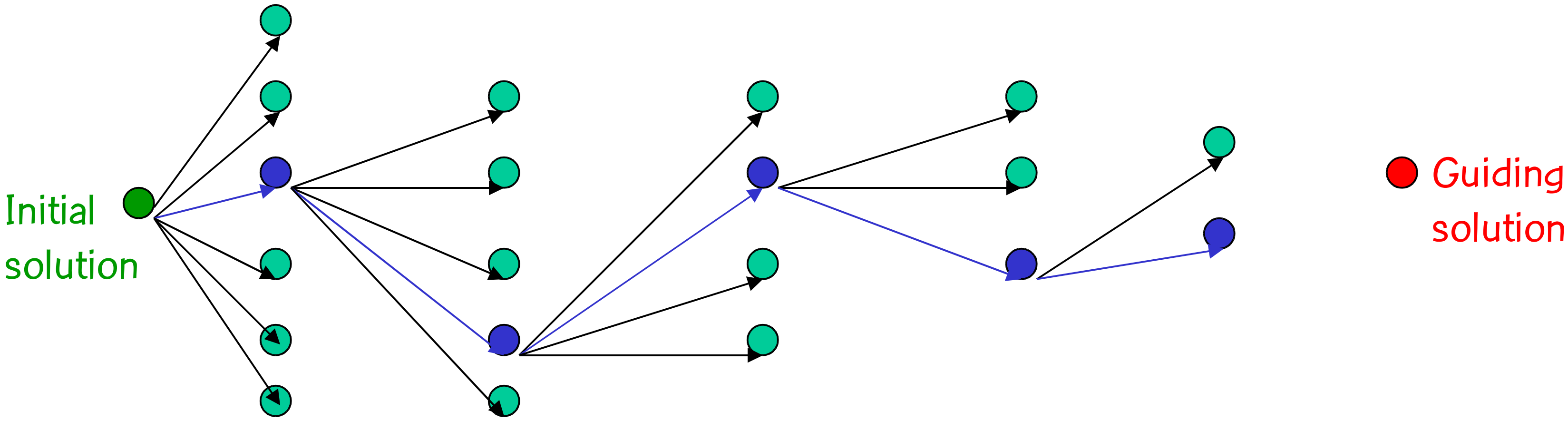
Path-relinking

- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are analyzed and best move is taken.



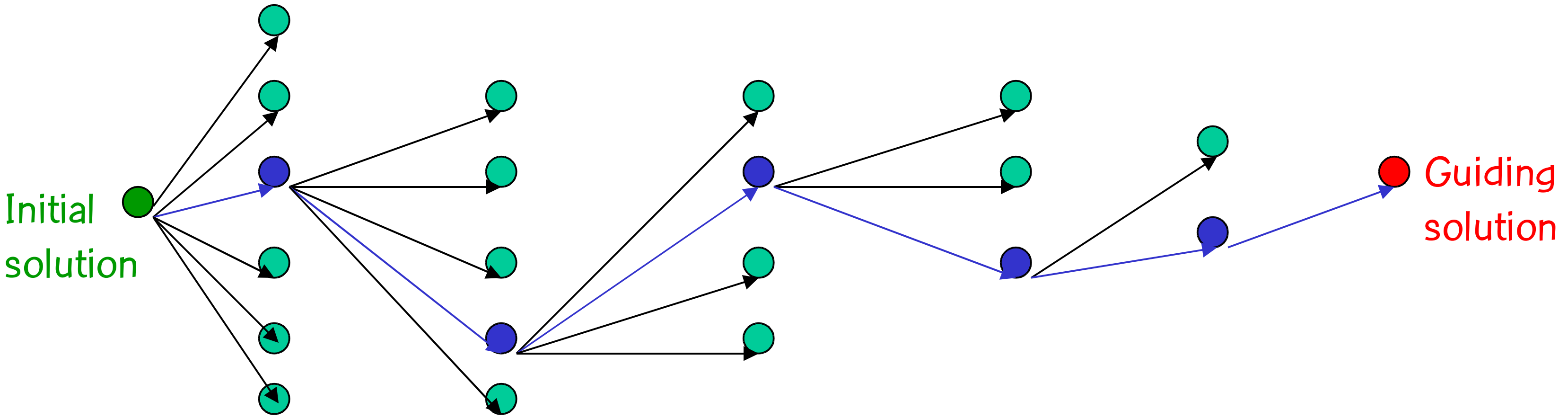
Path-relinking

- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are analyzed and best move is taken.



Path-relinking

- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are analyzed and best move is taken.



Output of PR usually is best solution in path.

Elite solutions

- To test whether a new solution should be inserted into the pool, we use a criterion based on symmetric difference between two solutions S_a and S_b: $| S_a \setminus S_b | + | S_b \setminus S_a |$
- A new solution is inserted only if its symmetric difference to each cheaper solution already there is at least four.
- Moreover, if pool is full, the new solution must also cost less than the most expensive element in the pool. In this case, the new solution replaces the one (among those with equal or greater cost) it is most similar to.

Elite solutions

- To test whether a new solution should be inserted into the pool, we use a criterion based on symmetric difference between two solutions S_a and S_b: $| S_a \setminus S_b | + | S_b \setminus S_a |$
- A new solution is inserted only if its symmetric difference to each cheaper solution already there is at least four.
- Moreover, if pool is full, the new solution must also cost less than the most expensive element in the pool. In this case, the new solution replaces the one (among those with equal or greater cost) it is most similar to.

Elite solutions

- To test whether a new solution should be inserted into the pool, we use a criterion based on symmetric difference between two solutions S_a and S_b: $|S_a \setminus S_b| + |S_b \setminus S_a|$
- A new solution is inserted only if its symmetric difference to each cheaper solution already there is at least four.
- Moreover, if pool is full, the new solution must also cost less than the most expensive element in the pool. In this case, the new solution replaces the one (among those with equal or greater cost) it is most similar to.

Intensification

- After each iteration, the solution S obtained by the local search is combined with a solution S' obtained from the pool.
- Solution S' is chosen at random, with probability proportional to its symmetric difference to S:
  - This tends to lead to longer paths on which to search

Intensification

- After each iteration, the solution S obtained by the local search is combined with a solution S' obtained from the pool.
- Solution S' is chosen at random, with probability proportional to its symmetric difference to S:
  - This tends to lead to longer paths on which to search

Post-optimization

Evolutionary path-relinking

a) Start with pool found at end of multistart phase: P_0 ; Set $k = 0$;

b) Combine with path-relinking all pairs of solutions in pool P_k ;

c) Solutions obtained by combining solutions in P_k are added to a new pool P_{k+1} following same constraints for updates as before;

d) If best solution of P_{k+1} is better than best solution of P_k , then set $k = k + 1$, and go to step (b);

Post-optimization

Evolutionary path-relinking

a) Start with pool found at end of multistart phase: P_0 ; Set $k = 0$;

b) Combine with path-relinking all pairs of solutions in pool P_k ;

c) Solutions obtained by combining solutions in P_k are added to a new pool P_{k+1} following same constraints for updates as before;

d) If best solution of P_{k+1} is better than best solution of P_k , then set $k = k + 1$, and go to step (b);

Post-optimization

Evolutionary path-relinking

a) Start with pool found at end of multistart phase: P_0 ; Set $k = 0$;

b) Combine with path-relinking all pairs of solutions in pool P_k ;

c) Solutions obtained by combining solutions in P_k are added to a new pool P_{k+1} following same constraints for updates as before;

d) If best solution of P_{k+1} is better than best solution of P_k , then set $k = k + 1$, and go to step (b);

Post-optimization

Evolutionary path-relinking

a) Start with pool found at end of multistart phase: P_0 ; Set $k = 0$;

b) Combine with path-relinking all pairs of solutions in pool P_k ;

c) Solutions obtained by combining solutions in P_k are added to a new pool P_{k+1} following same constraints for updates as before;

d) If best solution of P_{k+1} is better than best solution of P_k , then set $k = k + 1$, and go to step (b);

Parameters

- Besides random number seed, HYBRID takes only two input parameters:
  - N: number of iterations
  - E: size of pool of elite solutions
- In standard version, we use $N = 32$ and $E = 10$.

Parameters

- Besides random number seed, HYBRID takes only two input parameters:
  - N: number of iterations
  - E: size of pool of elite solutions
- In standard version, we use N = 32 and E = 10.

Parameters

- Recall that running time of multistart phase depends linearly on number of iterations N, whereas post-optimization depends (roughly) quadratically on the pool size E.
- Therefore, if we want to multiply the average running time of the algorithm by some factor X, we just multiply N by X and E by sqrt(X), rounding off appropriately.

Parameters

- Recall that running time of multistart phase depends linearly on number of iterations N, whereas post-optimization depends (roughly) quadratically on the pool size E.
- Therefore, if we want to multiply the average running time of the algorithm by some factor X, we just multiply N by X and E by sqrt(X), rounding off appropriately.

Empirical results

Experimental setup

- Algorithm implemented in C++ and compiled with the SGI MIPSPro C++ compiler (v. 7.30) with flags –O3 –OPT:Olimit=6586
- Runs were done on an SGI Challenge with 28 196-MHz MIPS 10000 processors, but each execution was limited to a single processor
- All CPU times reported are measured by the getrusage function with a precision of 1/60 second
- Random number generator: Mersenne Twister (Matsumoto and Nishimura, 1998)

Empirical results

Experimental setup

- Algorithm implemented in C++ and compiled with the SGI MIPSPro C++ compiler (v. 7.30) with flags –O3 –OPT:Olimit=6586
- Runs were done on an SGI Challenge with 28 196-MHz MIPS 10000 processors, but each execution was limited to a single processor
- All CPU times reported are measured by the getrusage function with a precision of 1/60 second
- Random number generator: Mersenne Twister (Matsumoto and Nishimura, 1998)

Empirical results

Experimental setup

- Algorithm implemented in C++ and compiled with the SGI MIPSPro C++ compiler (v. 7.30) with flags –O3 –OPT:Olimit=6586
- Runs were done on an SGI Challenge with 28 196-MHz MIPS 10000 processors, but each execution was limited to a single processor
- All CPU times reported are measured by the getrusage function with a precision of 1/60 second
- Random number generator: Mersenne Twister (Matsumoto and Nishimura, 1998)

Empirical results

Experimental setup

- Algorithm implemented in C++ and compiled with the SGI MIPSPro C++ compiler (v. 7.30) with flags –O3 –OPT:Olimit=6586
- Runs were done on an SGI Challenge with 28 196-MHz MIPS R10000 processors, but each execution was limited to a single processor
- All CPU times reported are measured by the getrusage function with a precision of 1/60 second
- Random number generator: Mersenne Twister (Matsumoto and Nishimura, 1998)

Empirical results

Test problems

- Algorithm was tested on all classes from UflLib (Hoefer, 2003) and on class GHOSH, described in Ghosh (2003).
- In every case, the number of users and potential facilities is the same (locations are the same).

`http://www.mpi-sb.mpg.de/units/ag1/projects/benchmarks/UflLib`

Empirical results

Test problems

- Algorithm was tested on all classes from UflLib (Hoefer, 2003) and on class GHOSH, described in Ghosh (2003).
- In every case, the number of users and potential facilities is the same (locations are the same).

http://www.mpi-sb.mpg.de/units/ag1/projects/benchmarks/UflLib

Instance class	Reference	Instances/Size	Notes
BK	Bilde & Krarup (1977)	200 instances, 30 to 100 users	d ~ [0,1000] c ≥ 1000
FPP	Kochetov (2003)	80 instances, 133 & 307 users	Meant to be challenging for algorithms based on local search.
GAP	Kochetov (2003)	120 instances, 100 users	Large duality gaps. Hard for dual-based methods.
GHOSH	Ghosh (2003)	90 instances, 250, 500, & 750 users	d ~ [1000,2000] A: c ~ [100,200] B: c ~ [1000,2000] C: c ~ [10K,20K]

Test problems

BK used in Hoefer's comparative analysis.

Instance class	Reference	Instances/Size	Notes
GR	Galvão & Raggi (1989)	50 instances, 50 to 200 users	d ~ shortest paths given as matrices
M*	Kratica et al. (2001)	22 instances, 100 to 2000 users	Meant to be close to real-life applications: many near-optimal solutions.
MED	Ahn et al. (1998); Barahona & Chudak (1999)	18 instances, 500 to 3000 users	Random points in unit square, Euclidean distances with 4 signif. digits.
ORLIB	Beasley (1993)	15 instances, 50 to 1000 users	Instances originally proposed for capacitated facility location problems.

Test problems

GR, M*, MED, and ORLIB used in Hoefer's comparative analysis.

Empirical results

Quality assessment

- Standard version of algorithm
- Run ten times on each instance with ten random number seeds (1,...,10)
- Compare to optima for FPP, GAP, BK, GR, and ORLIB and best upper bounds for MED and M*
- Geometric means given for times.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- On all five classes in Hoefer's analysis, our algorithms does very well.
  - Matches best known bounds (usually optima) on GR, M*, and ORLIB.
  - Few unlucky runs on class BK.
  - On MED, solutions were on average 0.4% better than best known bounds
  - Did well on GHOSH, compared to two algorithms.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- On all five classes in Hoefer's analysis, our algorithms does very well.
- Matches best known bounds (usually optima) on GR, M*, and ORLIB.
- Few unlucky runs on class BK.
- On MED, solutions were on average 0.4% better than best known bounds
- Did well on GHOSH, compared to two algorithms.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- On all five classes in Hoefer's analysis, our algorithms does very well.
- Matches best known bounds (usually optima) on GR, M*, and ORLIB.
- Few unlucky runs on class BK.
- On MED, solutions were on average 0.4% better than best known bounds
- Did well on GHOSH, compared to two algorithms.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- On all five classes in Hoefer's analysis, our algorithms does very well.
- Matches best known bounds (usually optima) on GR, M*, and ORLIB.
- Few unlucky runs on class BK.
- On MED, solutions were on average 0.4% better than best known bounds
- Did well on GHOSH, compared to two algorithms.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- On all five classes in Hoefer's analysis, our algorithms does very well.
- Matches best known bounds (usually optima) on GR, M*, and ORLIB.
- Few unlucky runs on class BK.
- On MED, solutions were on average 0.4% better than best known bounds
- Did well on GHOSH, compared to two algorithms.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- The remaining two classes: FPP & GAP were created with the intent of being hard.
- Solutions are much worse than for other classes.
- However, we show later that, if given more time, our algorithm can do well on these classes, too.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- The remaining two classes: FPP & GAP were created with the intent of being hard.
- Solutions are much worse than for other classes.
- However, we show later that, if given more time, our algorithm can do well on these classes, too.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Quality assessment

- The remaining two classes: FPP & GAP were created with the intent of being hard.
- Solutions are much worse than for other classes.
- However, we show later that, if given more time, our algorithm can do well on these classes, too.

Class	Avg % dev	Time (secs)
BK	0.001	0.28
FPP	27.999	7.36
GAP	5.935	1.63
GHOSH	(0.039)	30.66
GR	0.000	0.31
M*	0.000	7.45
MED	(0.392)	284.88
ORLIB	0.000	0.18

Empirical results

Comparative analysis

- We have seen that our algorithm produces very good quality solutions on most of the classes of instances tested.
- On there own, however, these results don't mean much.
- Any reasonably scalable algorithm, given enough time, should be able to find good solutions.
- With this in mind: we compare our algorithm with the best algorithm from Hoefer's analysis: the tabu search of Michel and Van Hentenryck (2003)

Empirical results

Comparative analysis

- We have seen that our algorithm produces very good quality solutions on most of the classes of instances tested.
- On their own, however, these results don't mean much.
- Any reasonably scalable algorithm, given enough time, should be able to find good solutions.
- With this in mind: we compare our algorithm with the best algorithm from Hoefer's analysis: the tabu search of Michel and Van Hentenryck (2003)

Empirical results

Comparative analysis

- We have seen that our algorithm produces very good quality solutions on most of the classes of instances tested.
- On there own, however, these results don't mean much.
- Any reasonably scalable algorithm, given enough time, should be able to find good solutions.
- With this in mind: we compare our algorithm with the best algorithm from Hoefer's analysis: the tabu search of Michel and Van Hentenryck (2003)

Empirical results

Comparative analysis

- We have seen that our algorithm produces very good quality solutions on most of the classes of instances tested.
- On there own, however, these results don't mean much.
- Any reasonably scalable algorithm, given enough time, should be able to find good solutions.
- With this in mind: we compare our algorithm with the best algorithm from Hoefer's analysis: the tabu search of Michel and Van Hentenryck (2003)

- We downloaded TABU from UflLib and ran it on our computer with 500 iterations (as in Hoefer's experiments).
- Since TABU was faster than our standard version, we compare with a faster HYBRID with N = 8 and E = 5.
- Both algorithms were run 10 times on each instance

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

- We downloaded TABU from UflLib and ran it on our computer with 500 iterations (as in Hoefer's experiments).
- Since TABU was faster than our standard version, we compare with a faster HYBRID with N = 8 and E = 5.
- Both algorithms were run 10 times on each instance

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

- We downloaded TABU from UflLib and ran it on our computer with 500 iterations (as in Hoefer's experiments).
- Since TABU was faster than our standard version, we compare with a faster HYBRID with N = 8 and E = 5.
- Both algorithms were run 10 times on each instance

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

- Both algorithms had similar running times.
- Even though running times are much lower than for standard version of HYBRID, both algorithms find very good quality solutions on five classes in Hoefer's analysis.
- On classes FPP, GAP, & MED, however, HYBRID does better than TABU.
- Time spent on classes FPP and GAP is only about one second.

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

- Both algorithms had similar running times.
- Even though running times are much lower than for standard version of HYBRID, both algorithms find very good quality solutions on five classes in Hoefer's analysis.
- On classes FPP, GAP, & MED, however, HYBRID does better than TABU.
- Time spent on classes FPP and GAP is only about one second.

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

- Both algorithms had similar running times.
- Even though running times are much lower than for standard version of HYBRID, both algorithms find very good quality solutions on five classes in Hoefer's anaylsis.
- On classes FPP, GAP, & MED, however, HYBRID does better than TABU.
- Time spent on classes FPP and GAP is only about one second.

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

- Both algorithms had similar running times.
- Even though running times are much lower than for standard version of HYBRID, both algorithms find very good quality solutions on five classes in Hoefer's analysis.
- On classes FPP, GAP, & MED, however, HYBRID does better than TABU.
- Time spent on classes FPP and GAP is only about one second.

Class	HYBRID		TABU	
	%dev	time	%dev	time
BK	.028	0.082	0.076	0.152
FPP	66.49	1.730	97.06	0.604
GAP	9.502	0.369	16.50	0.244
GHOSH	(0.032)	7.887	0.002	4.621
GR	0.000	0.087	0.103	0.158
M*	0.004	2.087	0.011	1.615
MED	(0.369)	75.231	0.073	69.552
ORLIB	0.000	0.046	0.024	0.155

time in seconds (196 MHz R10000)

Longer runs

- Both HYBRID and TABU should benefit if given more time to solve instances in GAP and FPP.
- We ran TABU with 1000, 2000, 4000, ..., 64000 iterations and HYBRID with N:E pairs 4:3, 8:5, 16:7, 32:10 (standard HYBRID), 64:14, 128:20, 256:28, and 512:40.

Longer runs

- Both HYBRID and TABU should benefit if given more time to solve instances in GAP and FPP.
- We ran TABU with 1000, 2000, 4000, ..., 64000 iterations and HYBRID with N:E pairs 4:3, 8:5, 16:7, 32:10 (standard HYBRID), 64:14, 128:20, 256:28, and 512:40.

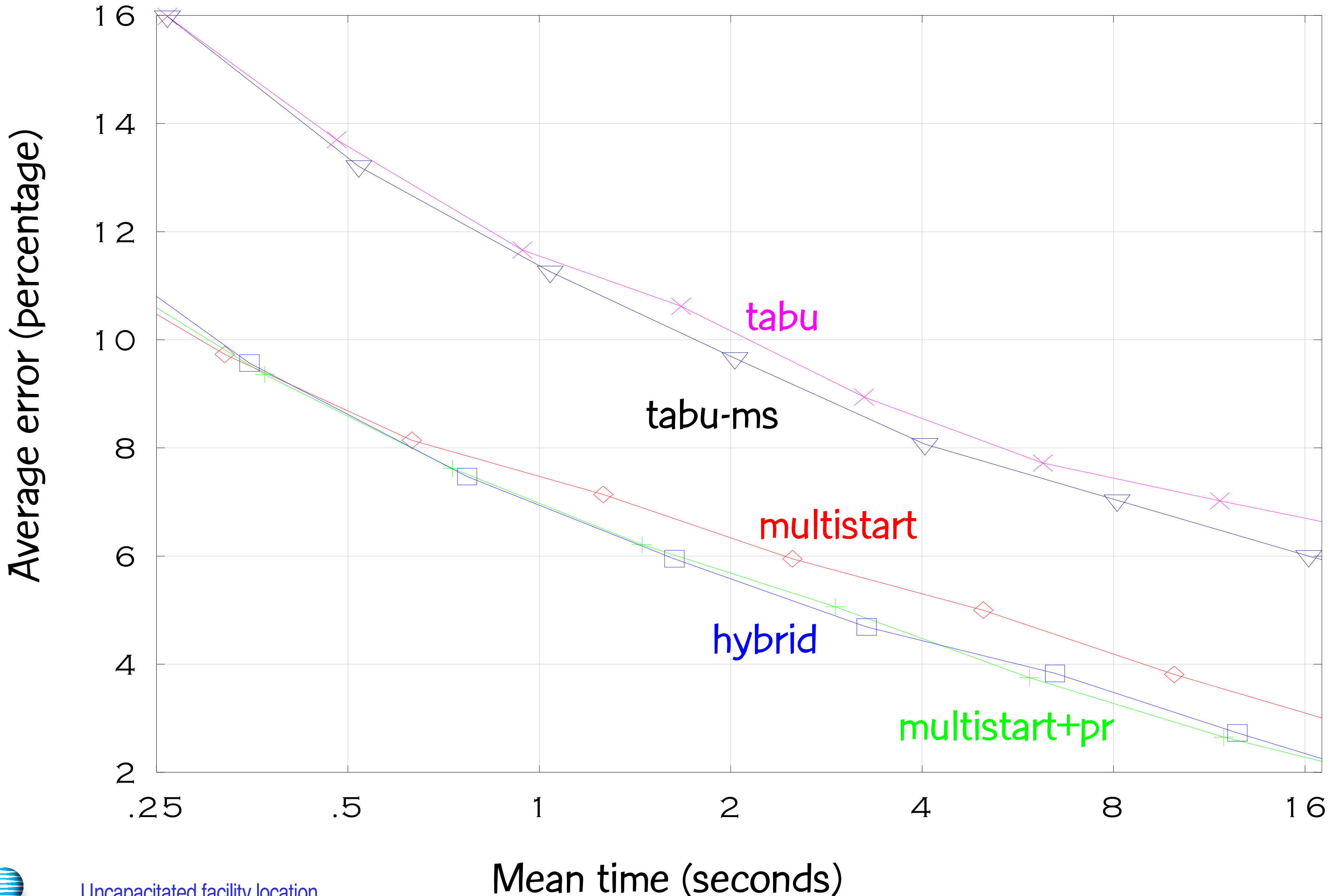
HYBRID				TABU		
iterations	elite	% error	time	iterations	% error	time
4	3	12.961	0.14	500	16.50	0.25
8	5	9.543	0.37	1000	14.38	0.46
16	7	7.407	0.78	2000	12.40	0.88
32	10	5.932	1.63	4000	10.62	0.88
64	14	4.561	3.23	8000	8.94	3.27
128	20	3.541	6.49	16000	7.72	6.24
256	28	2.700	12.54	32000	7.02	11.85
512	40	1.685	24.69	64000	6.35	22.62

Means over ten runs.

GAP class

Time in seconds (196MHz R10000)

GAP class



16
14
12
10
8
6
4
2
Average error (percentage)
.25
.5
1
2
4
8
16
Mean time (seconds)
tabu
tabu-ms
multistart
hybrid
multistart+pr

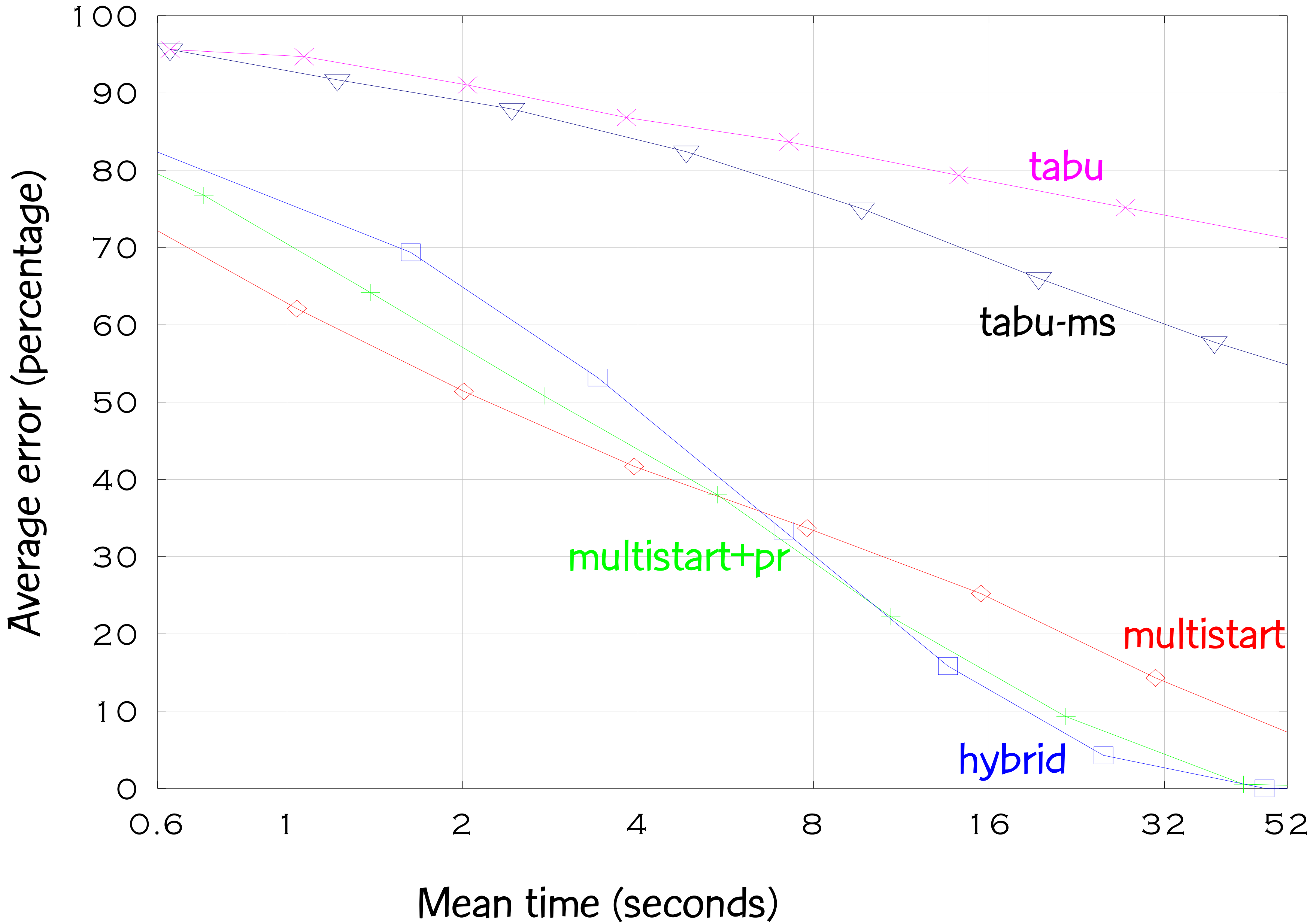
HYBRID				TABU		
iterations	elite	% error	time	iterations	% error	time
4	3	82.832	0.58	500	97.06	0.60
8	5	65.265	1.59	1000	94.22	1.04
16	7	48.413	3.49	2000	91.14	1.97
32	10	27.610	7.15	4000	86.81	3.86
64	14	13.279	13.79	8000	83.67	7.34
128	20	2.307	25.33	16000	79.32	14.34
256	28	0.018	48.17	32000	75.16	27.71
512	40	0.009	93.59	64000	71.15	52.60

Means over ten runs.

FPP class

Time in seconds (196MHz R10000)

FPP class



Uncapacitated facility location

AT&T

Paper

M.G.C. Resende and R.F. Werneck, A hybrid multi-start heuristic for the uncapacitated facility location problem, AT&T Labs Research Technical Report TD-5RELRR, Florham Park, NJ, Sept. 2003, revised Nov. 2004. To appear in European J. of Operational Research.

http://www.research.att.com/~mgcr/doc/guflp.pdf

Software availability

Our software (local search, and hybrid heuristics for p-median and facility location) as well as all test instances used in our studies are available for download at:

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