

# Advanced School and Workshop on Mathematical Techniques and Problems in Telecommunications

Tomar, Portugal  
September 8-12, 2003



## Short course: Some applications of combinatorial optimization in telecommunications

**Maurício G.C. RESENDE**

AT&T Labs Research  
Florham Park, New Jersey, USA

# Combinatorial Optimization

Handbook of Applied Optimization

P.M. Pardalos and M.G.C. Resende, eds. Oxford U. Press, 2002

**Combinatorial optimization:** process of finding the best, or optimal, solution for problems with a discrete set of feasible solutions.

**Applications:** e.g. routing, scheduling, packing, inventory and production management, location, logic, and assignment of resources.

**Economic impact:** e.g. transportation (airlines, trucking, rail, and shipping), forestry, manufacturing, logistics, aerospace, energy (electrical power, petroleum, and natural gas), agriculture, biotechnology, financial services, and **telecommunications**.

# Combinatorial Optimization

- Given:
  - discrete set of solutions  $X$
  - objective function  $f(x): x \in X \rightarrow \mathbb{R}$
- Objective:
  - find  $x \in X : f(x) \leq f(y), \forall y \in X$

# Combinatorial Optimization

- Much progress in recent years on finding exact (provably optimal) solution: dynamic programming, cutting planes, branch and cut, ...
- Many hard combinatorial optimization problems are still not solved exactly and require good heuristic methods.
- Aim of heuristic methods for combinatorial optimization is to quickly produce good-quality solutions, without necessarily providing any guarantee of solution quality.

# Metaheuristics

Metaheuristics: Computer Decision-Making

M.G.C. Resende and J.P. de Sousa, eds., Kluwer, 2003

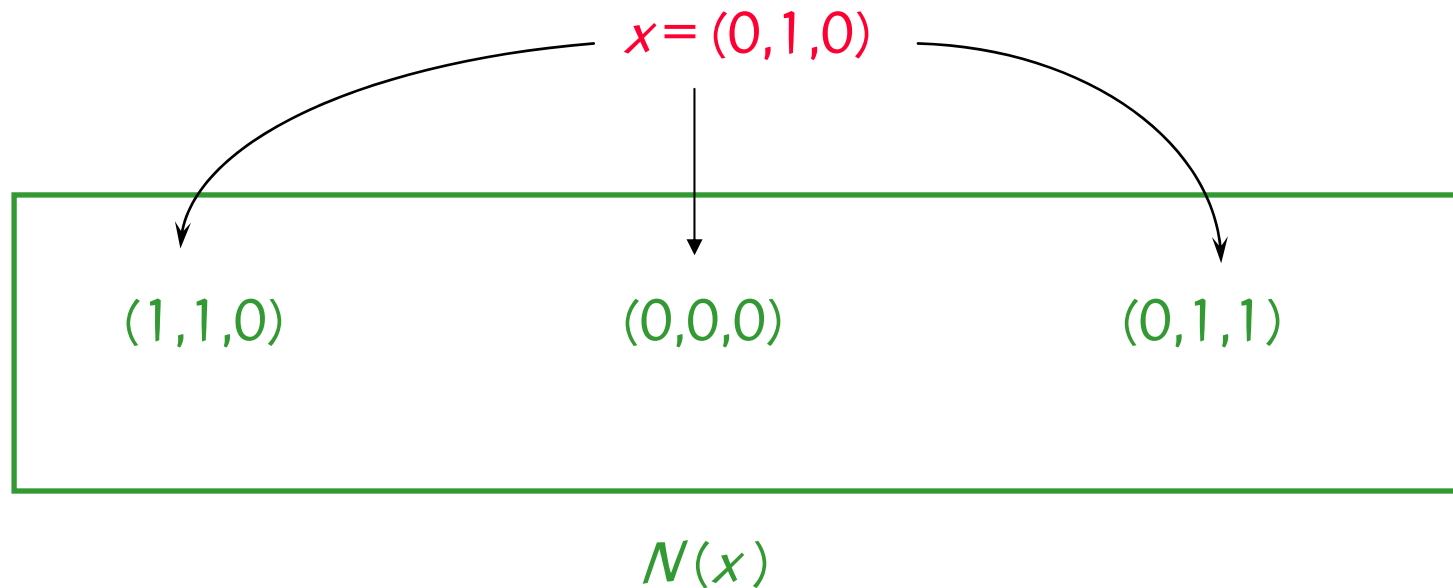
- **Metaheuristics** are high level procedures that coordinate simple heuristics, such as **local search**, to find solutions that are of better quality than those found by the simple heuristics alone.
- **Examples:** simulated annealing, genetic algorithms, tabu search, scatter search, ant colony optimization, variable neighborhood search, and **GRASP**.

# Local Search

- To define local search, one needs to specify a local neighborhood structure.
- Given a solution  $x$ , the elements of the neighborhood  $N(x)$  of  $x$  are those solutions  $y$  that can be obtained by applying an elementary modification (often called a move) to  $x$ .

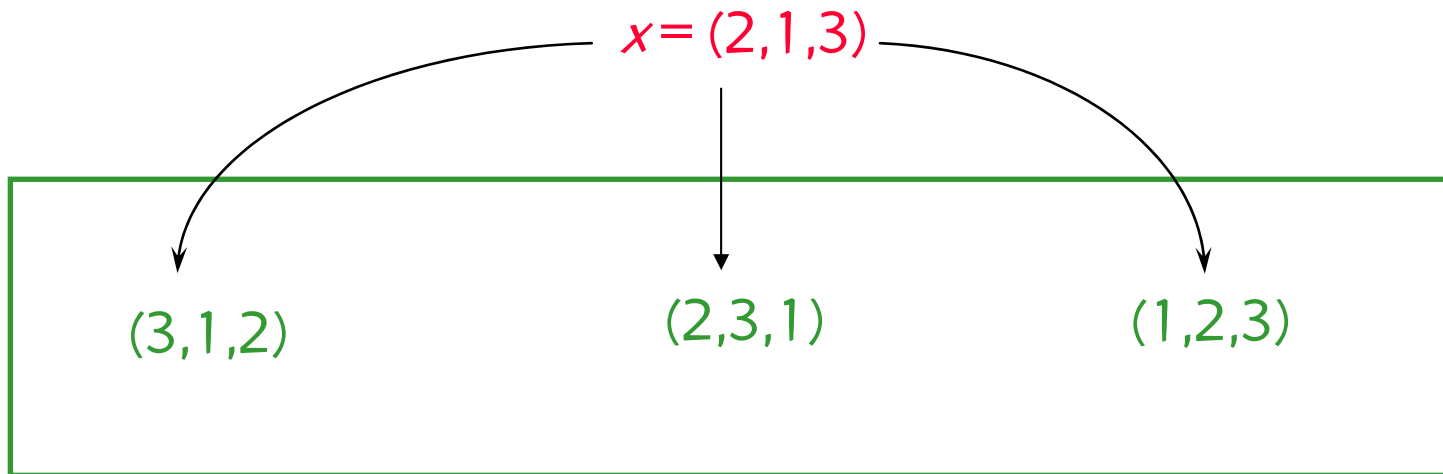
# Local Search Neighborhoods

Consider  $x = (0, 1, 0)$  and the 1-flip neighborhood of a 0/1 array.



# Local Search Neighborhoods

Consider  $x = (2, 1, 3)$  and the 2-swap neighborhood of a permutation array.





# Local Search

Given an initial solution  $x_0$ , a neighborhood  $N(x)$ , and function  $f(x)$  to be minimized:

$x = x_0 ;$

while (  $\exists y \in N(x) \mid f(y) < f(x)$  ) {

$x = y ;$  ← move to better  
                    solution  $y$

}

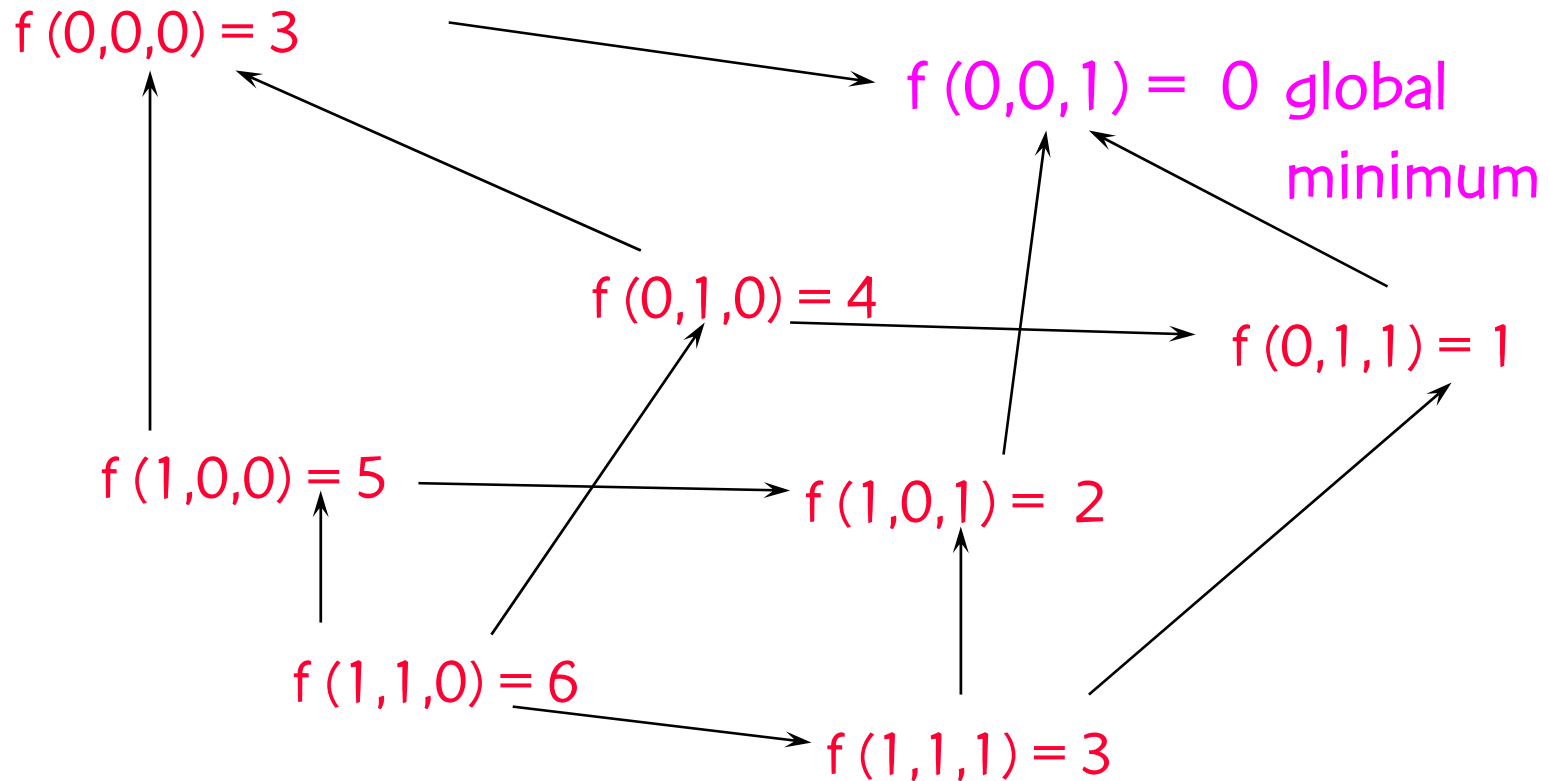
check for better solution in  
neighborhood of  $x$

At the end,  $x$  is a local minimum of  $f(x)$  .

Time complexity of local search  
can be exponential.

# Local Search

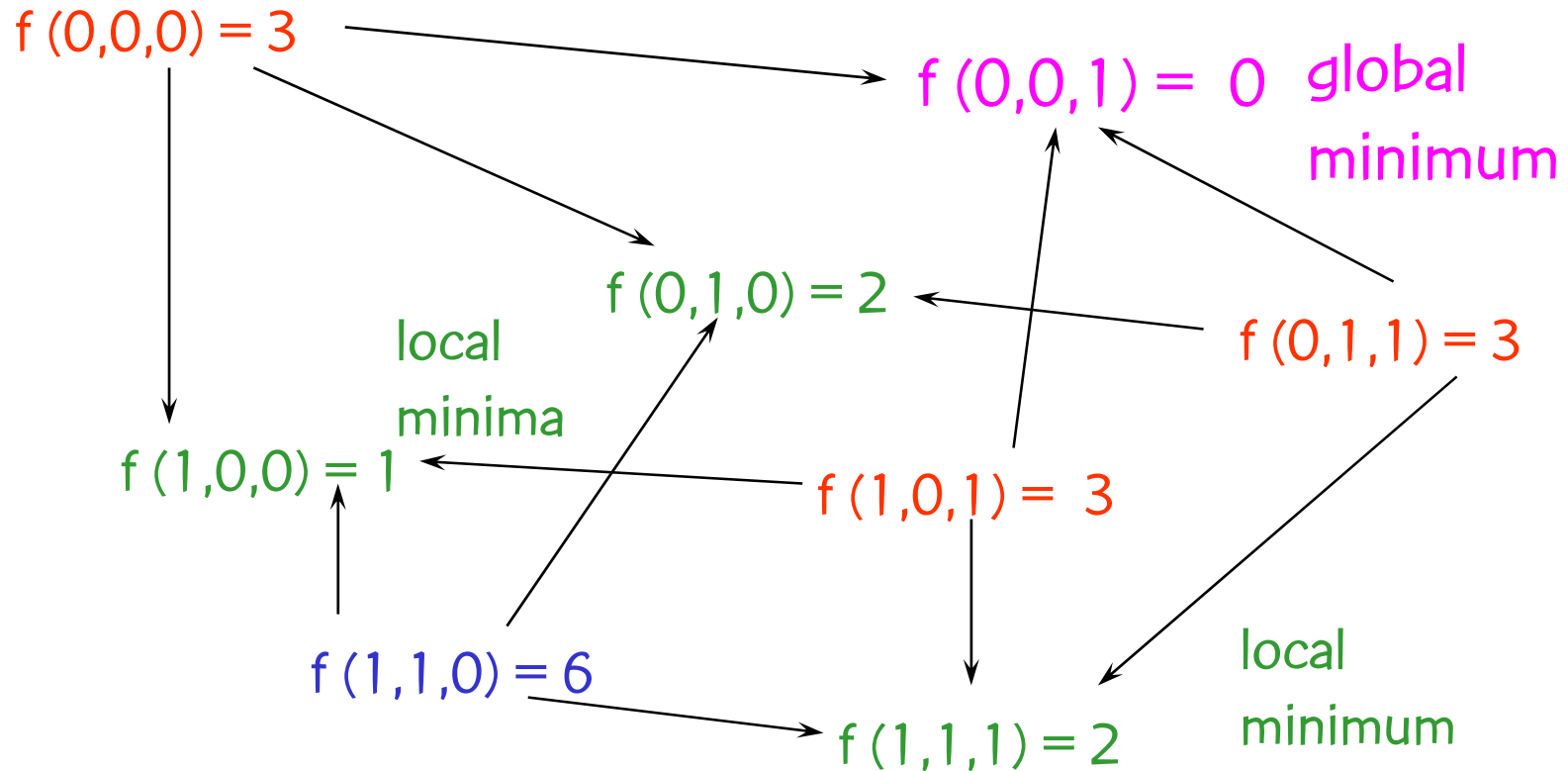
(ideal situation)



With any starting solution Local Search finds the global optimum.

# Local Search

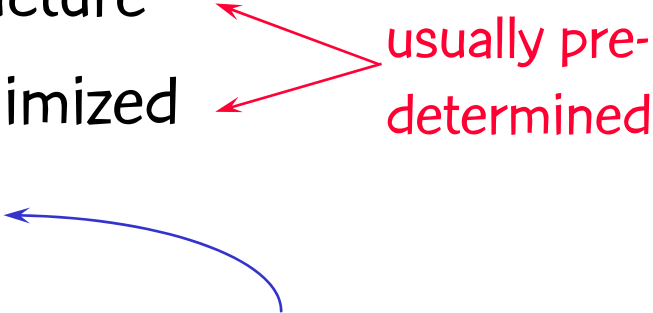
(more realistic situation)



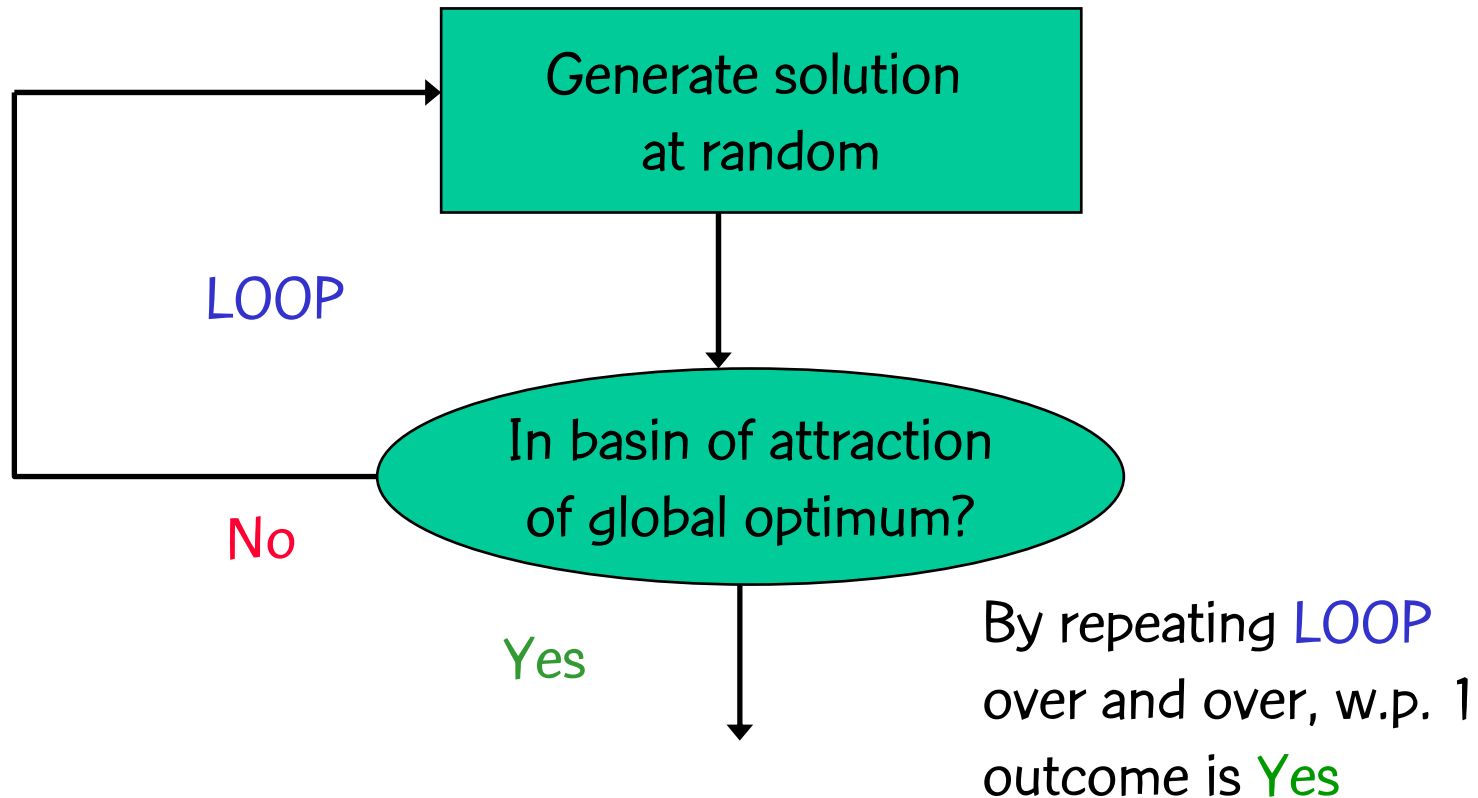
But some starting solutions lead Local Search to a local minimum.

# Local Search

Effectiveness of local search depends on several factors:

- neighborhood structure
  - function to be minimized
  - starting solution
- usually pre-determined
- usually easier to control
- 

# Local search with random starting solutions



Local search leads to global optimum.

# The greedy algorithm

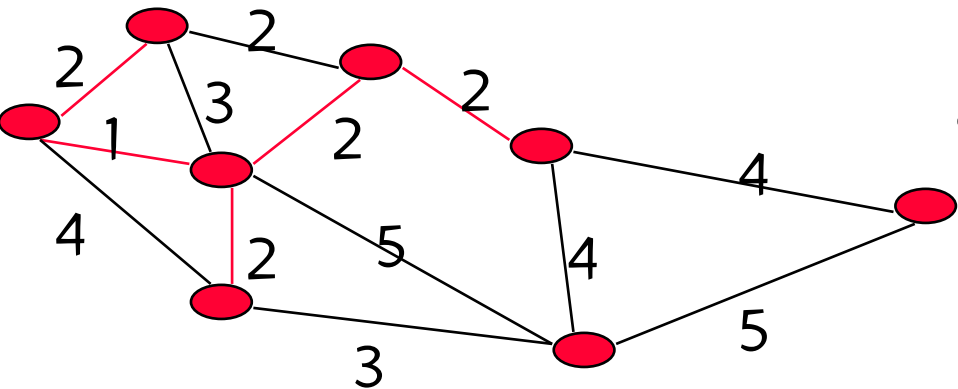
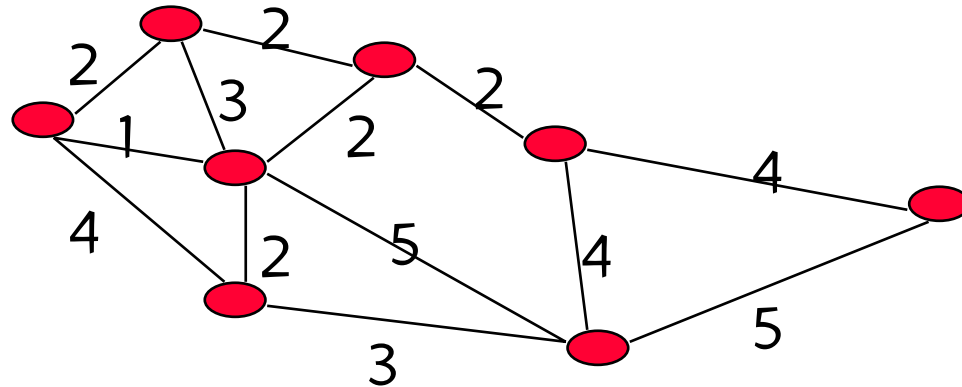
- To define a semi-greedy heuristic, we must first consider the greedy algorithm.
- Greedy algorithm: constructs a solution, one element at a time:

repeat until done

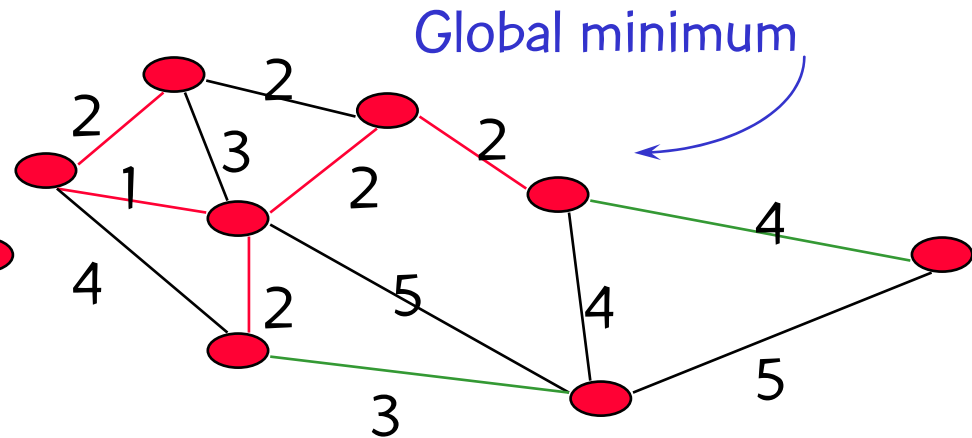
- Defines candidate elements.
- Applies a greedy function to each candidate element.
- Ranks elements according to greedy function value.
- Add best ranked element to solution.

# The greedy algorithm

## An example



Edges of weight 1 & 2



Edges of weight 3 & 4

Global minimum

# The greedy algorithm

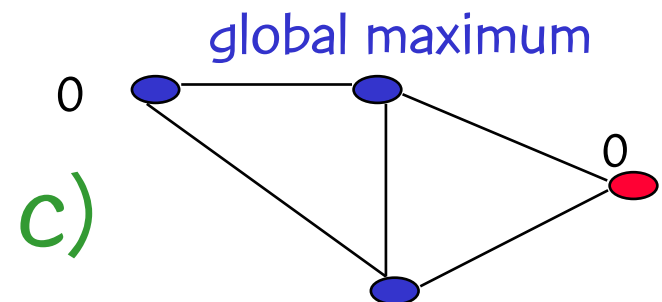
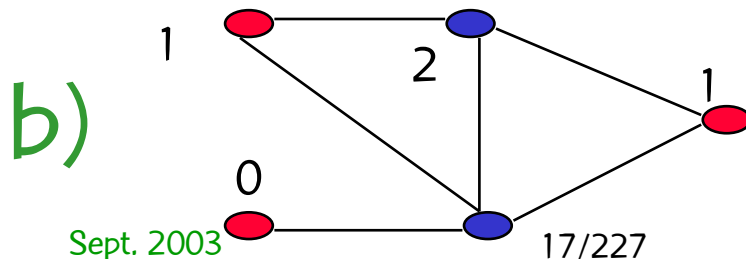
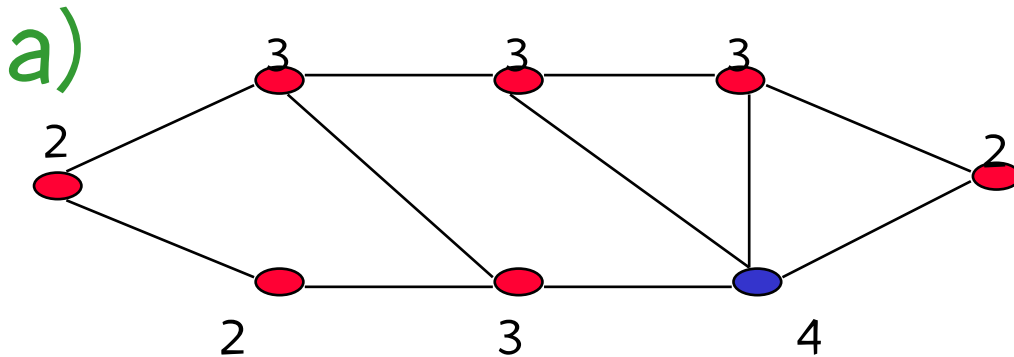
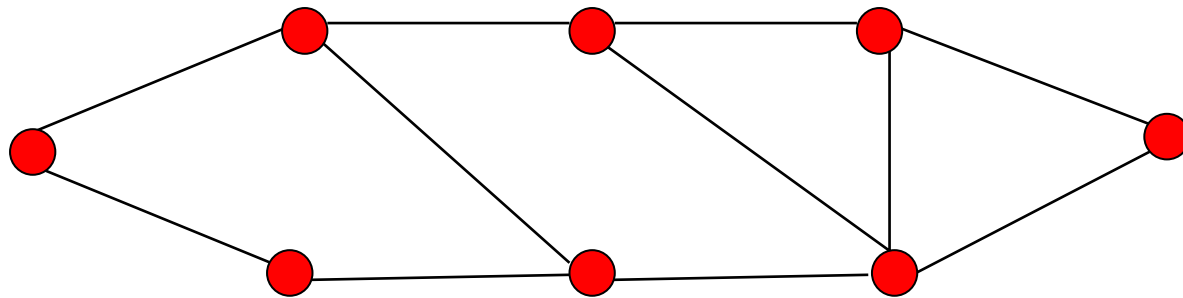
## Another example

- Maximum clique: Given graph  $G = (V, E)$ , find largest subgraph of  $G$  such that all vertices are mutually adjacent.
  - greedy algorithm builds solution, one element (vertex) at a time
  - candidate set: unselected vertices adjacent to all selected vertices
  - greedy function: vertex degree with respect to other candidate set vertices.



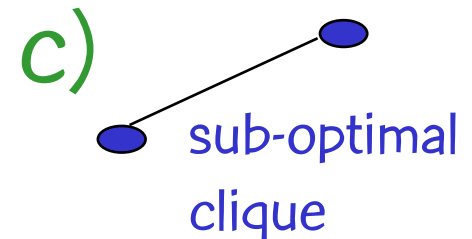
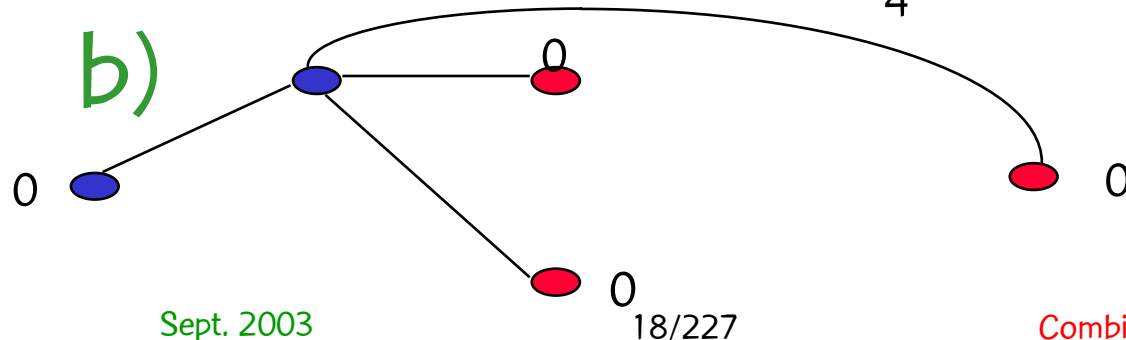
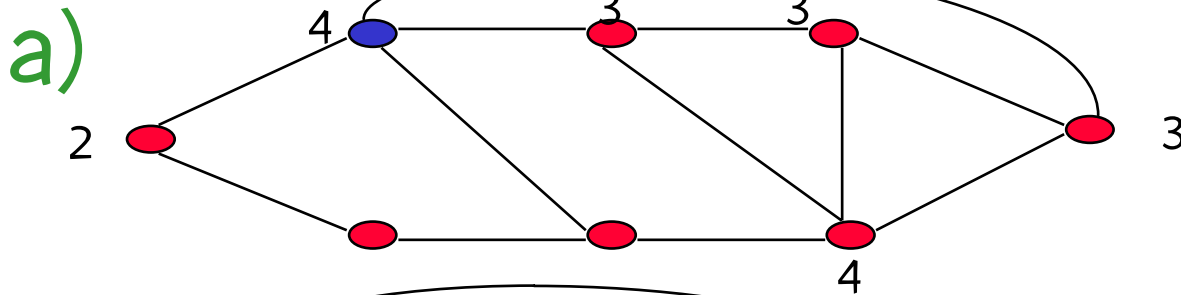
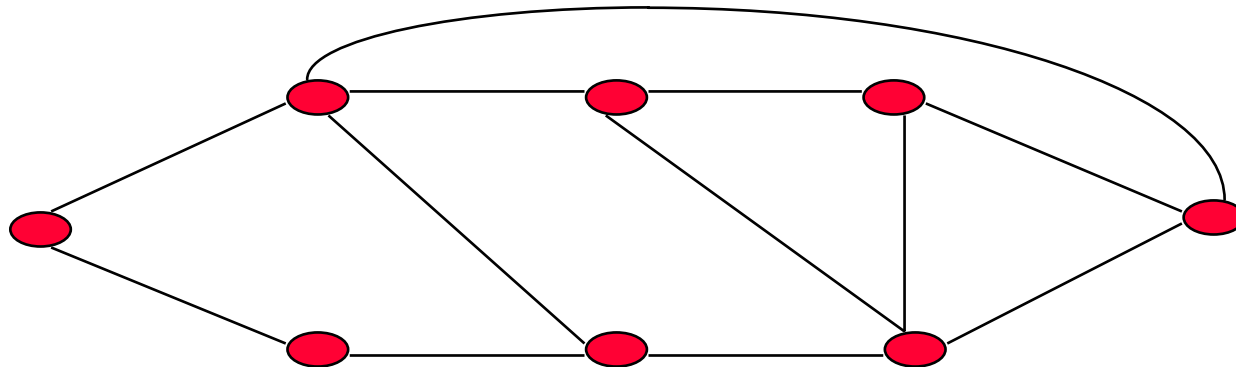
# The greedy algorithm

## Another example



# The greedy algorithm

## Another example

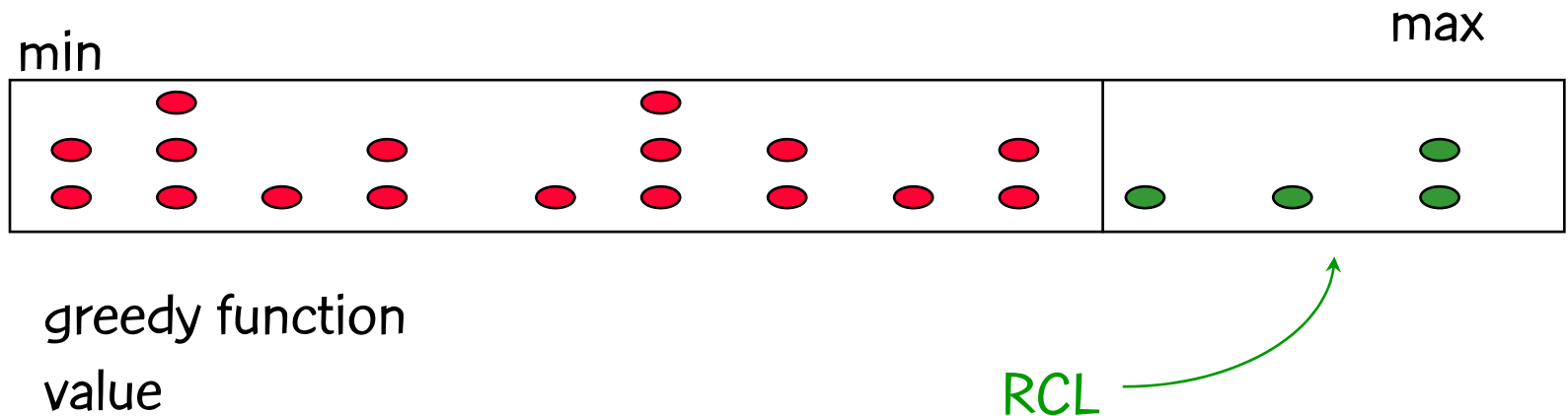


# Semi-greedy heuristic

- A semi-greedy heuristic tries to get around convergence to non-global local minima.
- 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

# Semi-greedy heuristic

Candidate elements are ranked according to greedy function value.



RCL is a set of well-ranked candidate elements.

# Semi-greedy heuristic

- Hart & Shogan (1987) propose two mechanisms for building the RCL:
  - Cardinality based: place  $k$  best candidates in RCL
  - Value based: place all candidates having greedy values better than  $\alpha \cdot \text{best\_value}$  in RCL, where  $\alpha \in [0,1]$ .
- Feo & Resende (1989) proposed semi-greedy construction, independently, as a basic component of GRASP.

# Hart-Shogan Algorithm

(maximization)

```
best_obj = 0;  
repeat many times{  
    x = semi-greedy_construction( );  
    if ( obj_function(x) > best_obj ){  
        x* = x;  
        best_obj = obj_function(x);  
    }  
}
```

# GRASP: Basic algorithm

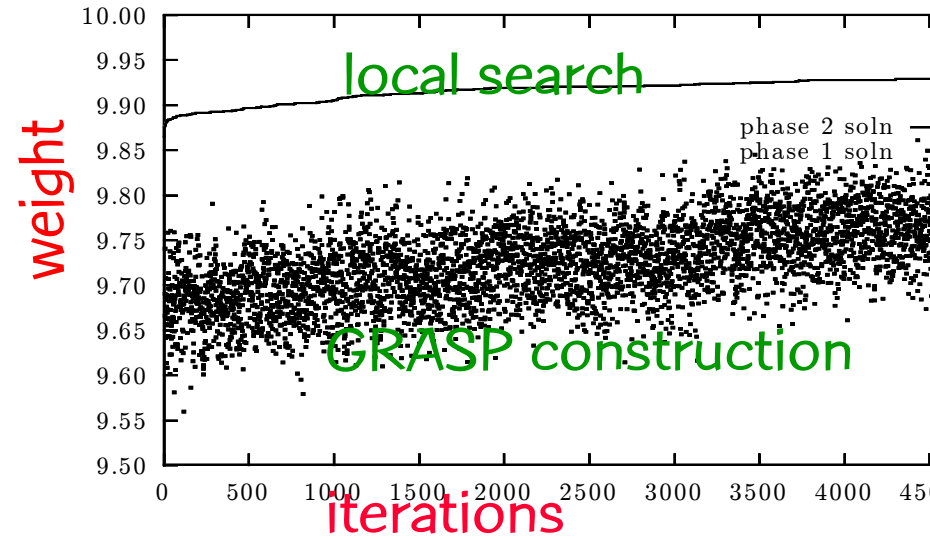
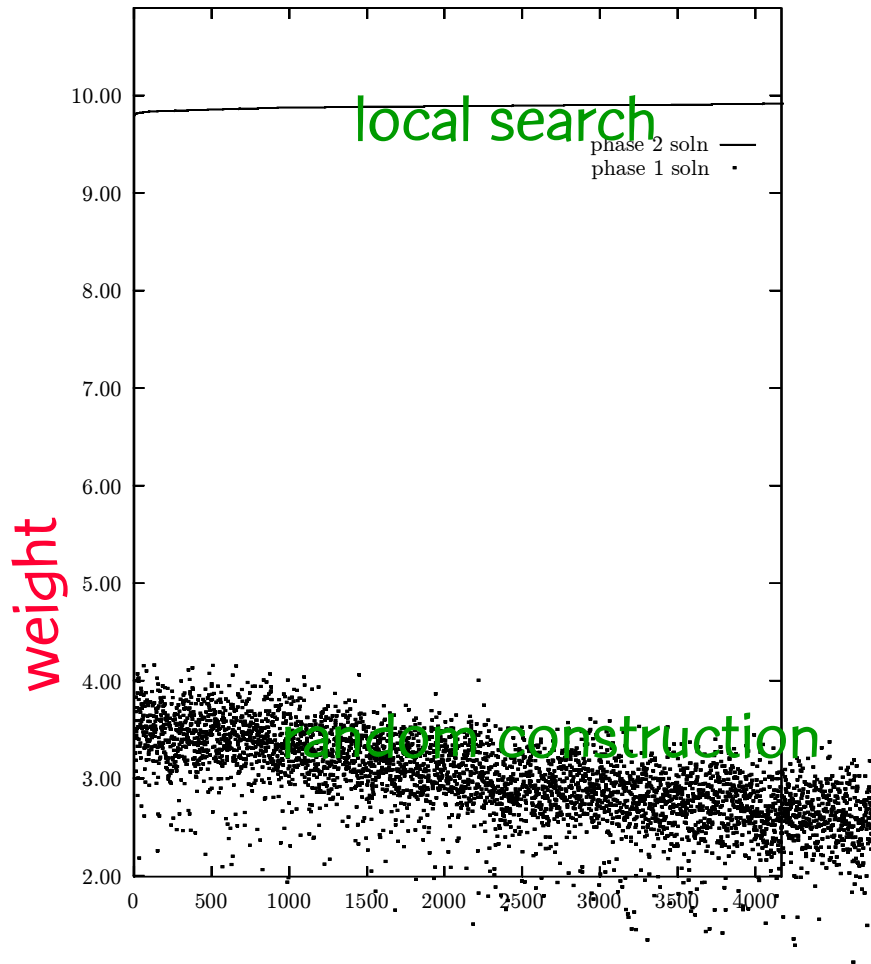
- GRASP:
  - Multistart metaheuristic:
    - Feo & Resende (1989): set covering
    - Feo & Resende (1995): first survey
    - Festa & Resende (2002): annotated bibliography
    - Resende & Ribeiro (2003): most recent survey
- Repeat for Max\_Iterations:
  - Construct a greedy randomized solution.
  - Use local search to improve the constructed solution.
  - Update the best solution found.

# GRASP: Basic algorithm

- Construction phase: greediness + randomization
  - Builds a feasible solution:
    - Use greediness to build restricted candidate list and apply randomness to select an element from the list.
    - Use randomness to build restricted candidate list and apply greediness to select an element from the list.
- Local search: search in the current neighborhood until a local optimum is found
  - Solutions generated by the construction procedure are not necessarily optimal:
    - Effectiveness of local search depends on: neighborhood structure, search strategy, and fast evaluation of neighbors, but also on the construction procedure itself.



# GRASP: Basic algorithm



Effectiveness of **greedy randomized**  
**purely randomized** construction

Application: modem placement  
max weighted covering problem  
maximization problem:  $\alpha = 0$

# Construction phase

- Greedy Randomized Construction:
  - Solution  $\leftarrow \emptyset$
  - Evaluate incremental costs of candidate elements
  - While Solution is not complete do:
    - Build restricted candidate list (RCL).
    - Select an element  $s$  from RCL at random.
    - Solution  $\leftarrow$  Solution  $\cup \{s\}$
    - Reevaluate the incremental costs.
  - endwhile

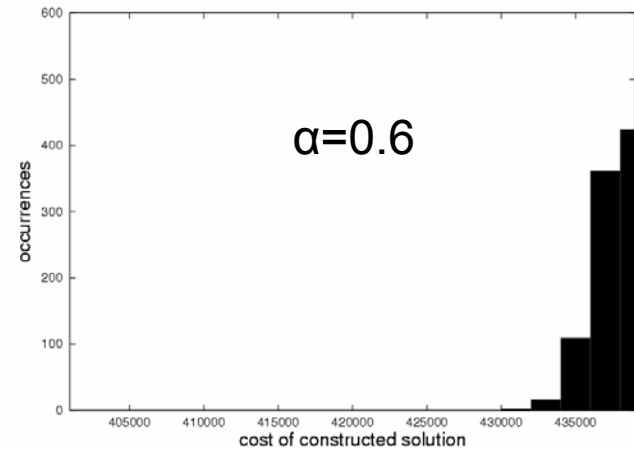
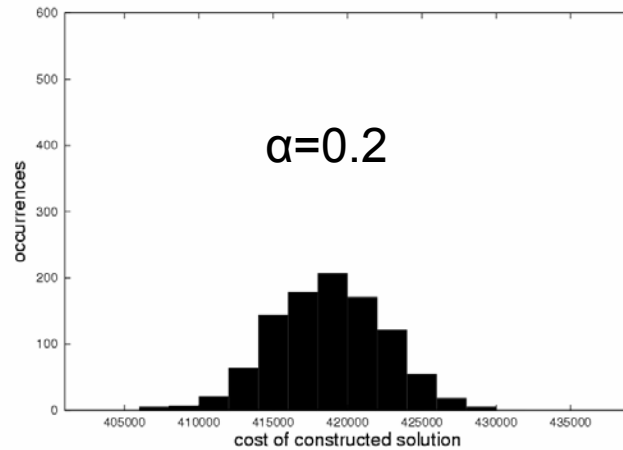
# Construction phase

- Minimization problem
- Basic construction procedure:
  - Greedy function  $c(e)$ : incremental cost associated with the incorporation of element  $e$  into the current partial solution under construction
  - $c^{\min}$  (resp.  $c^{\max}$ ): smallest (resp. largest) incremental cost
  - RCL made up by the elements with the smallest incremental costs.

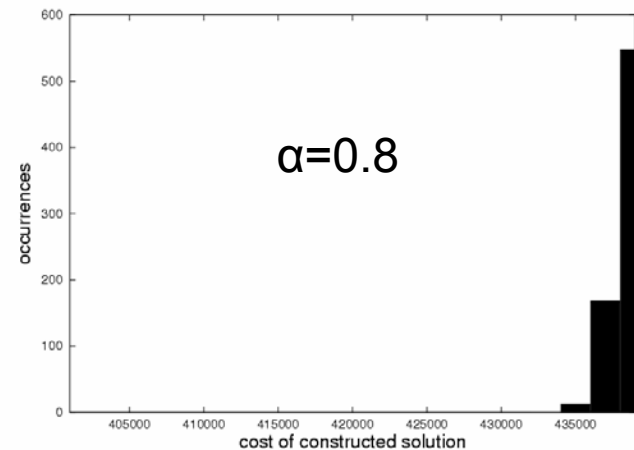
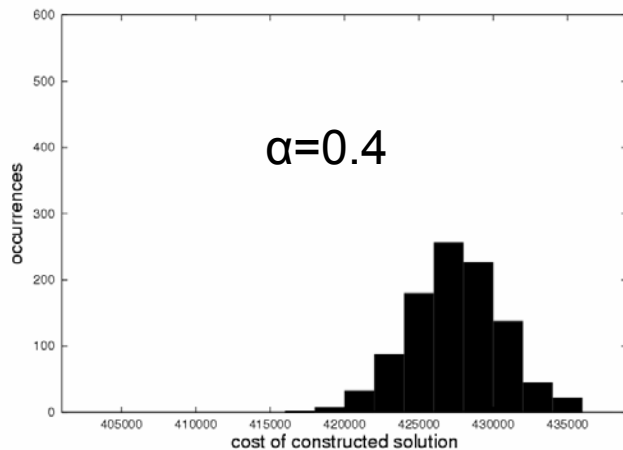
# Construction phase

- Cardinality-based construction:
  - $p$  elements with the smallest incremental costs
- Quality-based construction:
  - Parameter  $\alpha$  defines the quality of the elements in RCL.
  - RCL contains elements with incremental cost  $c^{\min} \leq c(e) \leq c^{\min} + \alpha (c^{\max} - c^{\min})$
  - $\alpha = 0$  : pure greedy construction
  - $\alpha = 1$  : pure randomized construction
- Select at random from RCL using uniform probability distribution

# Illustrative results: RCL parameter

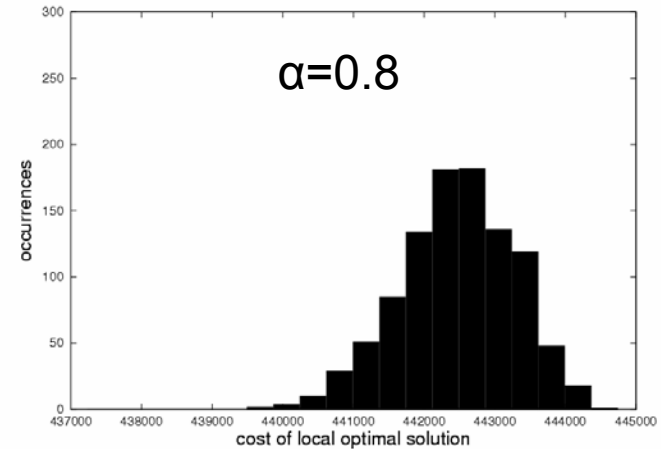
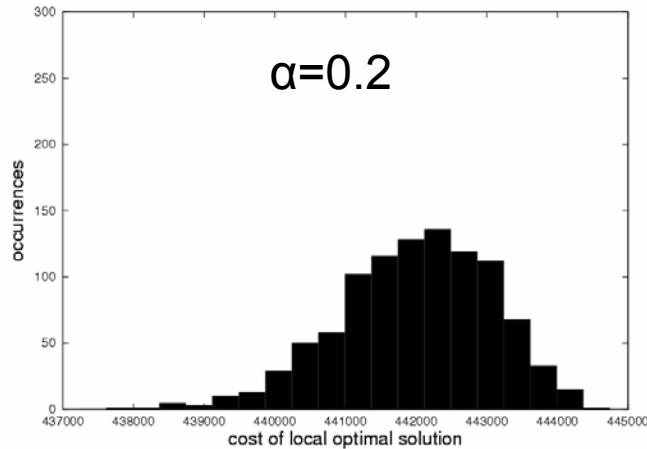


Construction phase only

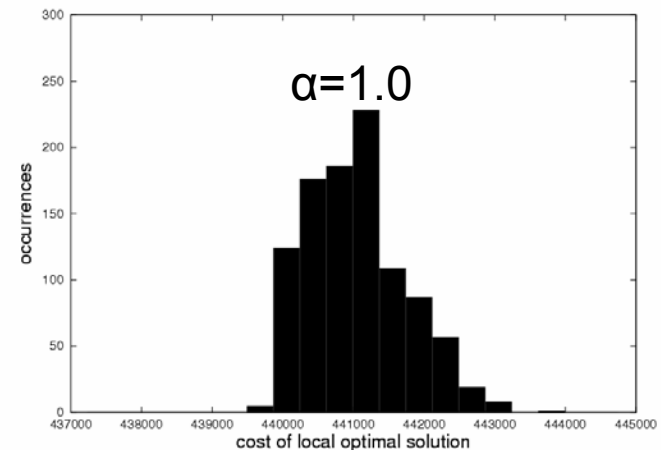
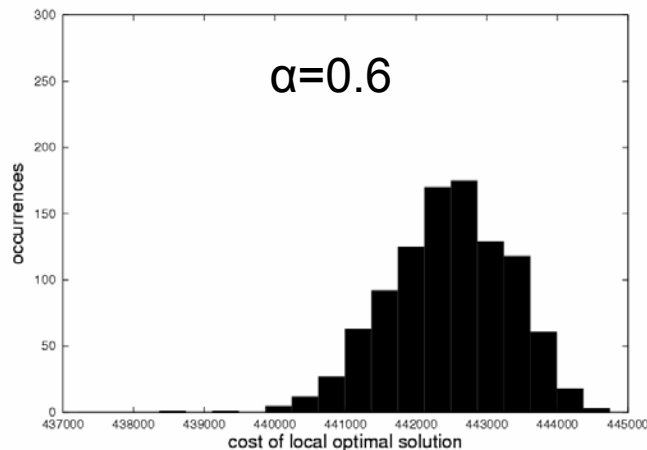


weighted MAX-SAT instance, 1000 GRASP iterations

# Illustrative results: RCL parameter

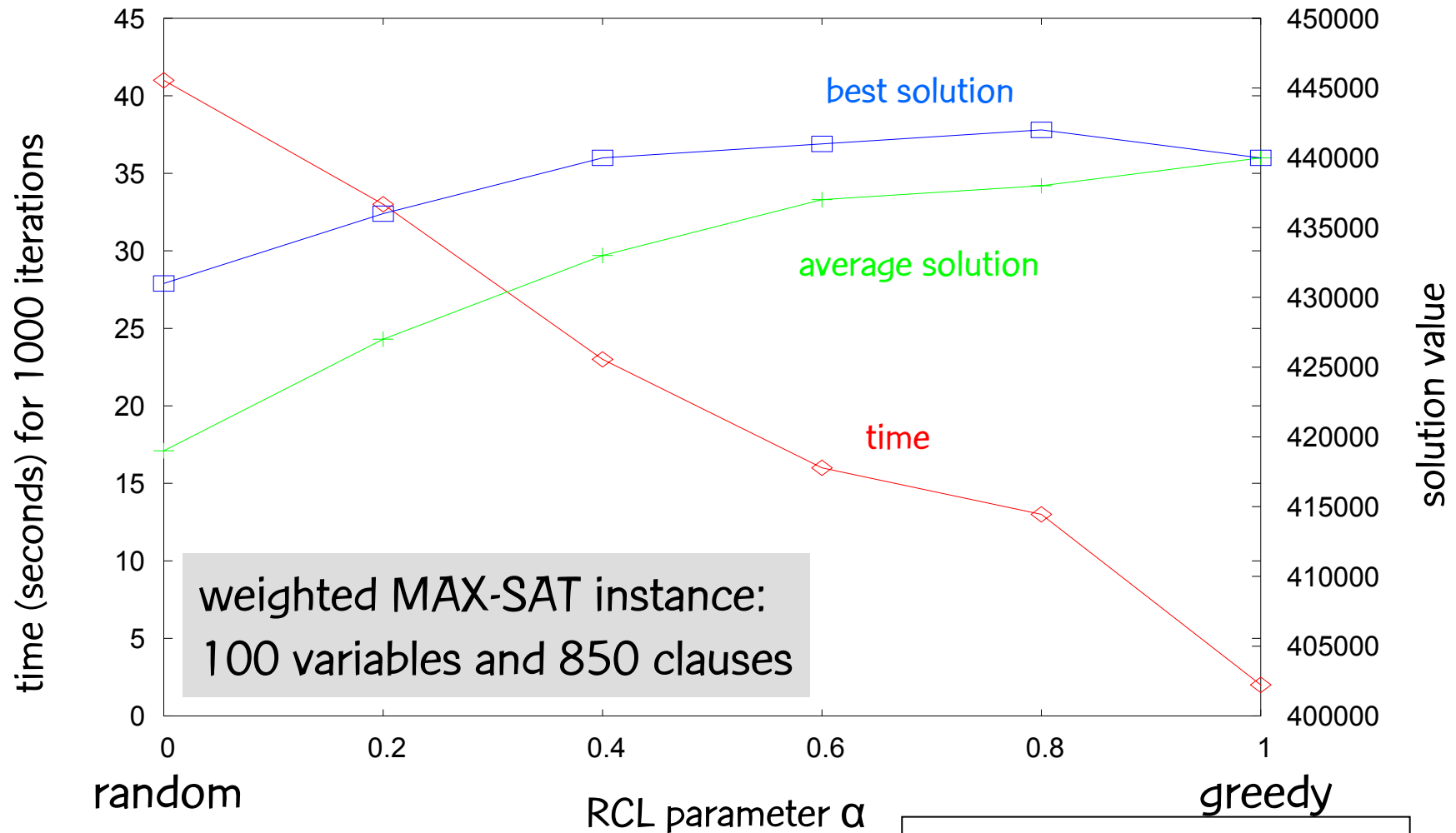


Construction + local search



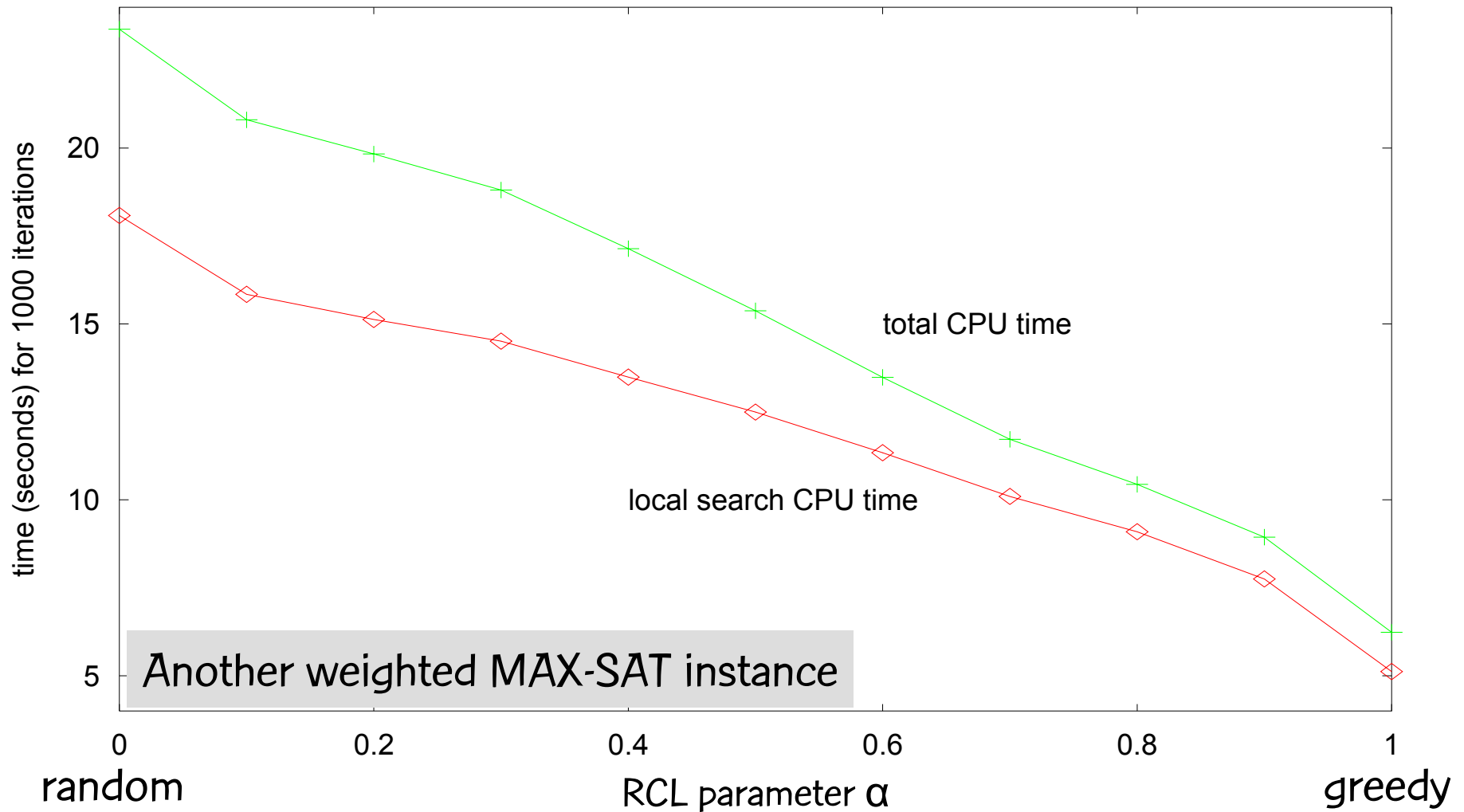
weighted MAX-SAT instance, 1000 GRASP iterations

# Illustrative results: RCL parameter



SGI Challenge 196 MHz

# Illustrative results: RCL parameter



SGI Challenge 196 MHz



# Enhanced construction strategies

- Reactive GRASP: Prais & Ribeiro (2000) (traffic assignment in TDMA satellites)
  - At each GRASP iteration, a value of the RCL parameter  $\alpha$  is chosen from a discrete set of values  $[\alpha_1, \alpha_2, \dots, \alpha_m]$ .
  - The probability that  $\alpha_k$  is selected is  $p_k$ .
  - **Reactive GRASP:** adaptively changes the probabilities  $[p_1, p_2, \dots, p_m]$  to favor values of  $\alpha$  that produce good solutions.
  - Other applications, e.g. to graph planarization, set covering, and weighted max-sat:
  - **Better solutions, at the cost of slightly larger times.**

# Enhanced construction strategies

- Cost perturbations: Canuto, Resende, & Ribeiro (2001) (prize-collecting Steiner tree)
  - Randomly perturb original costs and apply some heuristic.
  - Adds flexibility to algorithm design:
    - May be more effective than greedy randomized construction in circumstances where the construction algorithm is not very sensitive to randomization.
    - Also useful when no greedy algorithm is available.

# Enhanced construction strategies

- Sampled greedy: Resende & Werneck (2002) (p-median)
  - Randomly samples a small subset of candidate elements and selects element with smallest incremental cost.
- Random+greedy:
  - Randomly builds first part of the solution and completes the rest using pure greedy construction.

# Local search

- First improving vs. best improving:
  - First improving is usually faster.
  - Premature convergence to low quality local optimum is more likely to occur with best improving.
- **Variable Neighborhood Descent** (VND) to speedup search and to overcome optimality w.r.t. to simple (first) neighborhood: Ribeiro, Uchoa, & Werneck (2002) (Steiner problem in graphs)
- **Hashing** to avoid cycling or repeated application of local search to same solution built in the construction phase: Woodruff & Zemel (1993), Ribeiro et. al (1997) (query optimization), Martins et al. (2000) (Steiner problem in graphs)

# Local search

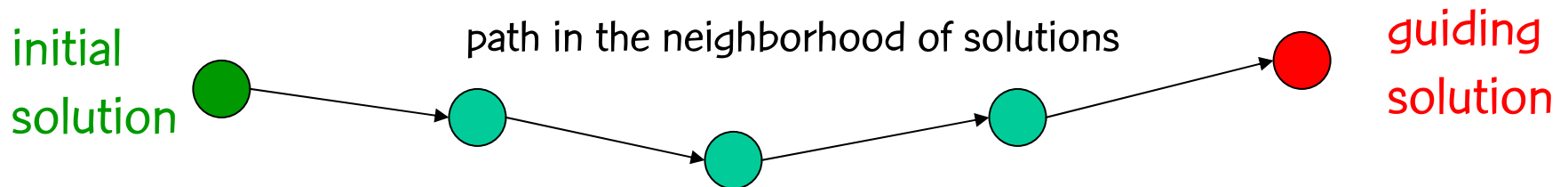
- **Filtering** to avoid application of local search to low quality solutions, only promising unvisited solutions are investigated: Feo, Resende, & Smith (1994), Prais & Ribeiro (2000) (traffic assignment), Martins et. al (2000) (Steiner problem in graphs)
- **Extended quick-tabu local search** to overcome premature convergence: Souza, Duhamel, & Ribeiro (2003) (capacitated minimum spanning tree, better solutions for largest benchmark problems)

# Path-relinking

- Path-relinking:
  - Intensification strategy exploring trajectories connecting elite solutions: Glover (1996)
  - Originally proposed in the context of tabu search and scatter search.
  - Paths in the solution space leading to other elite solutions are explored in the search for better solutions:
    - selection of moves that introduce attributes of the guiding solution into the current solution

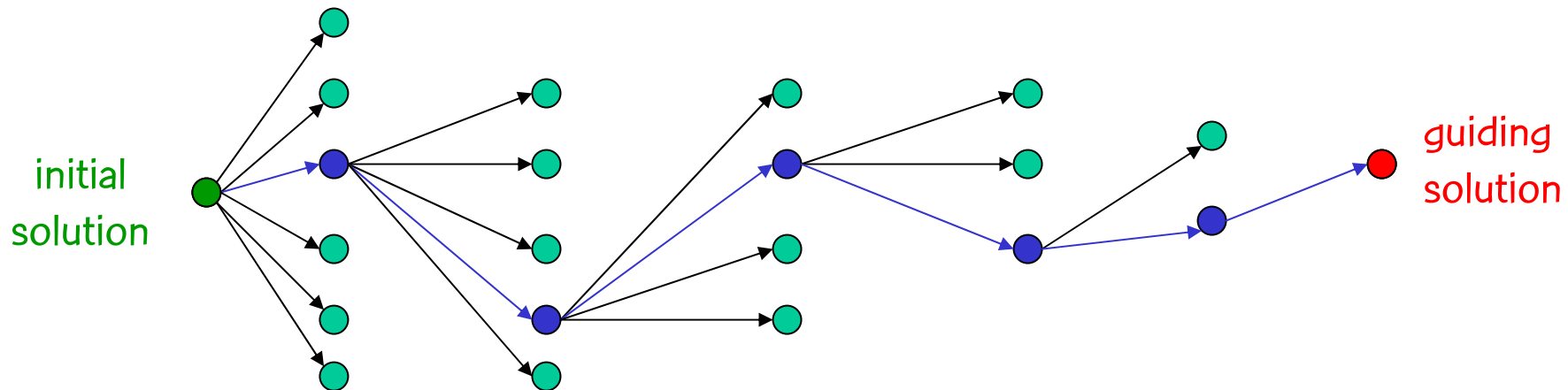
# Path-relinking

- Exploration of trajectories that connect high quality (elite) solutions:



# 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 evaluated and the best move is selected:





# Path-relinking

Solutions  $x$  and  $y$  to be combined.

$\Delta(x,y)$ : symmetric difference between  $x$  and  $y$

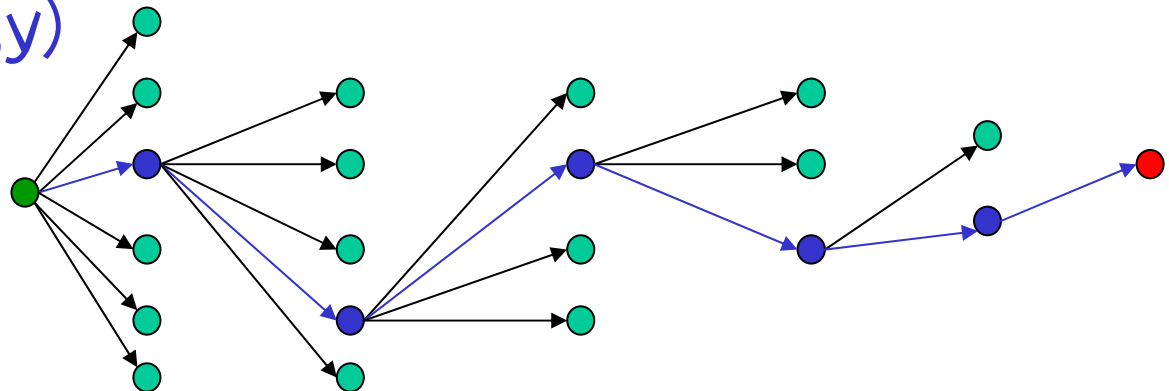
while (  $|\Delta(x,y)| > 0$  ) {

    evaluate moves corresponding in  $\Delta(x,y)$

    make best move

    update  $\Delta(x,y)$

}



# GRASP with path-relinking

- Originally used by Laguna and Martí (1999).
- Maintains a set of elite solutions found during GRASP iterations.
- After each GRASP iteration (construction and local search):
  - Use GRASP solution as **initial solution**.
  - Select an elite solution uniformly at random: **guiding solution** (may also be selected with probabilities proportional to the symmetric difference w.r.t. the initial solution).
  - Perform path-relinking between these two solutions.

# GRASP with path-relinking

- Repeat for Max\_Iterations:
  - Construct a greedy randomized solution.
  - Use local search to improve the constructed solution.
  - Apply path-relinking to further improve the solution.
  - Update the pool of elite solutions.
  - Update the best solution found.

# GRASP with path-relinking

- Variants: trade-offs between computation time and solution quality
  - Explore different trajectories (e.g. backward, forward): **better start from the best**, neighborhood of the initial solution is fully explored!
  - Explore both trajectories: **twice as much the time**, often with marginal improvements only!
  - Do not apply PR at every iteration, but instead only periodically: similar to **filtering** during local search.
  - Truncate the search, do not follow the full trajectory.
  - May also be applied as a **post-optimization** step to all pairs of elite solutions.

# GRASP with path-relinking

## Successful applications:

- Prize-collecting minimum Steiner tree problem: [Canuto, Resende, & Ribeiro \(2001\)](#) (e.g. improved all solutions found by approximation algorithm of Goemans & Williamson)
- Minimum Steiner tree problem: [Ribeiro, Uchoa, & Werneck \(2002\)](#) (e.g., best known results for open problems in series dv640 of the SteinLib)
- p-median: [Resende & Werneck \(2002\)](#) (e.g., best known solutions for problems in literature)

# GRASP with path-relinking

## Successful applications (cont'd):

- Capacitated minimum spanning tree: Souza, Duhamel, & Ribeiro (2002) (e.g., best known results for largest problems with 160 nodes)
- 2-path network design: Ribeiro & Rosseti (2002) (better solutions than greedy heuristic)
- Max-Cut: Festa, Pardalos, Resende, & Ribeiro (2002) (e.g., best known results for several instances)
- Quadratic assignment: Oliveira, Pardalos, & Resende (2003)

# GRASP with path-relinking

## Successful applications (cont'd):

- Job-shop scheduling: Aiex, Binato, & Resende (2003)
- Three-index assignment problem: Aiex, Resende, Pardalos, & Toraldo (2003)
- PVC routing: Resende & Ribeiro (2003)
- Phylogenetic trees: Ribeiro & Vianna (2003)
- Facility location: Resende & Werneck (2003) (e.g., best known solutions for problems in literature)

# GRASP with path-relinking

- $P$  is a set (pool) of elite solutions.
- Each iteration of first  $|P|$  GRASP iterations adds one solution to  $P$  (if different from others).
- After that: solution  $x$  is promoted to  $P$  if:
  - $x$  is better than best solution in  $P$ ,
  - $x$  is not better than best solution in  $P$ , but is better than worst and is sufficiently different from all solutions in  $P$ .



# Time-to-target-value plots

- **Proposition:** Let  $P(t,p)$  be the probability of not having found a given target solution value in  $t$  time units with  $p$  independent processors.

If  $P(t,1) = \exp[-(t-\mu)/\lambda]$  with non-negative  $\lambda$  and  $\mu$  (two-parameter exponential distribution), then  $P(t,p) = \exp[-p.(t-\mu)/\lambda]$ .

$\Rightarrow$  if  $p\mu \ll \lambda$ , then the probability of finding a solution within a given target value in time  $p.t$  with a sequential algorithm is approximately equal to that of finding a solution with the same quality in time  $t$  with  $p$  processors.

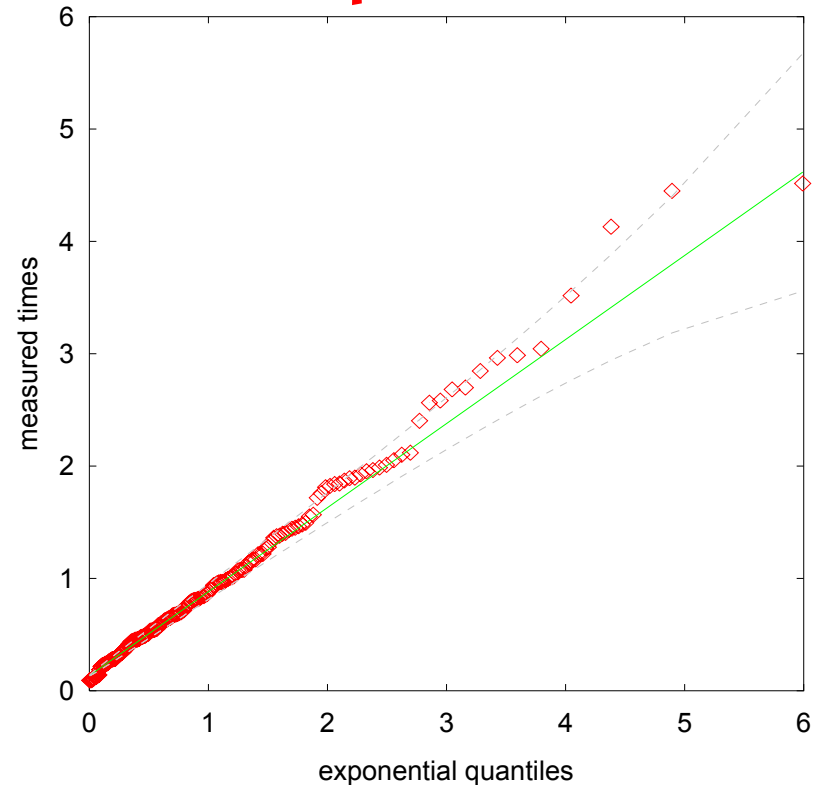
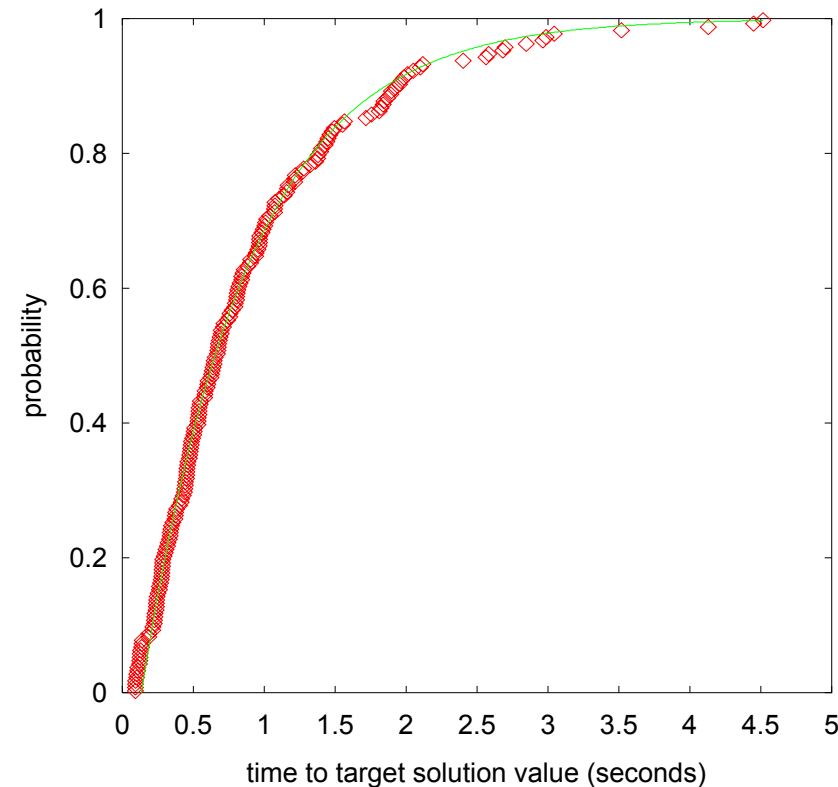
# Time-to-target-value plots

- Probability distribution of **time-to-target-solution-value**: Aiex, Resende, & Ribeiro (2002) and Aiex, Binato, & Resende (2003) have shown experimentally that both pure GRASP and GRASP with path-relinking present this behavior.

# Time-to-target-value plots

- Probability distribution of **time-to-target-solution-value**: experimental plots
- Select an instance and a target value.
- For each variant of GRASP with path-relinking:
  - Perform 200 runs using different seeds.
  - Stop when a solution value at least as good as the target is found.
  - For each run, measure the time-to-target-value.
  - Plot the probabilities of finding a solution at least as good as the target value within some computation time.

# Time-to-target-value plots



Random variable **time-to-target-solution value** fits a two-parameter exponential distribution.

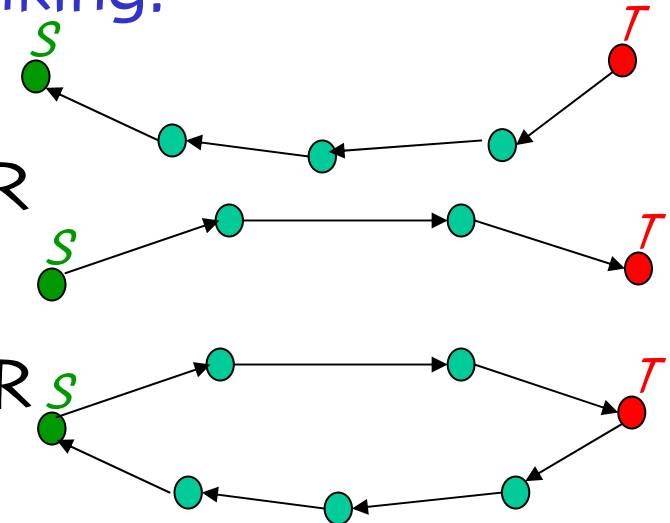
Therefore, one should expect approximate linear speedup in a straightforward (independent) parallel implementation.

# Variants of GRASP + PR

- Variants of GRASP with path-relinking:

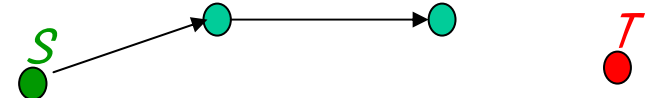
- GRASP: pure GRASP
- G+PR(B): GRASP with backward PR
- G+PR(F): GRASP with forward PR
- G+PR(BF): GRASP with two-way PR

*T*: elite solution    *S*: local search



- Other strategies:

- Truncated path-relinking
- Do not apply PR at every iteration (frequency)

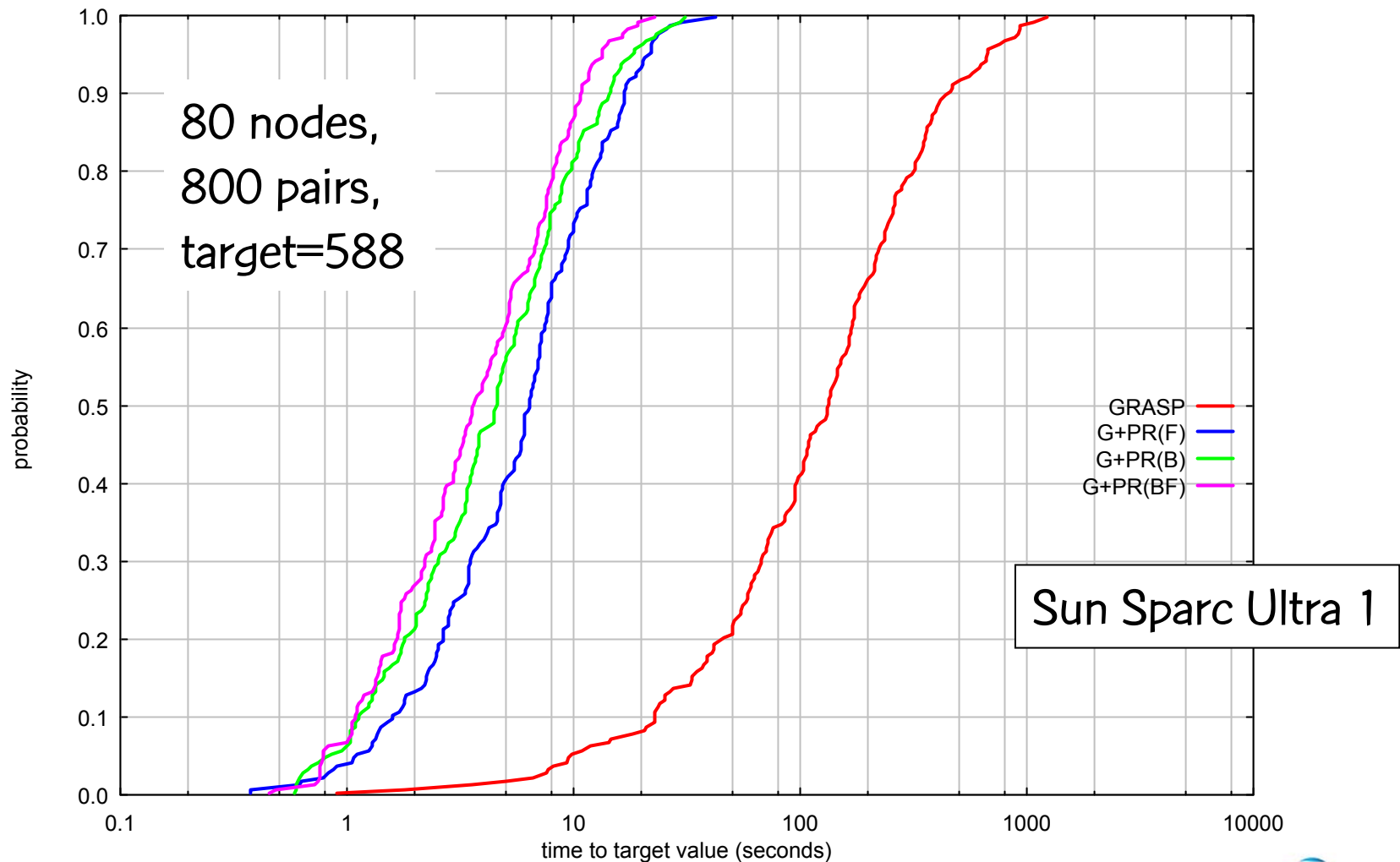


# 2-path network design problem

- 2-path network design problem:
  - Graph  $G=(V,E)$  with edge weights  $w_e$  and set  $D$  of origin-destination pairs (demands): find a minimum weighted subset of edges  $E' \subseteq E$  containing a 2-path (path with at most two edges) in  $G$  between the extremities of every origin-destination pair in  $D$ .
- Applications: design of communication networks, in which paths with few edges are sought to enforce high reliability and small delays

# 2-path network design problem

Each variant: 200 runs for one instance of 2PNDP



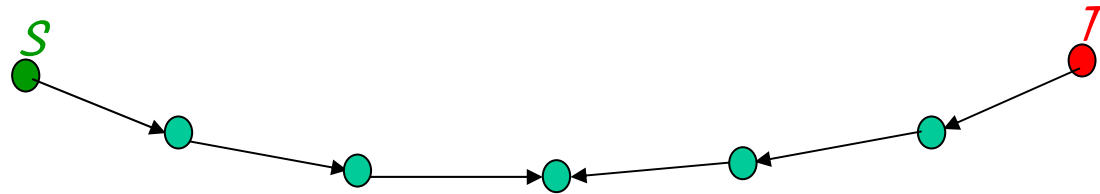
# 2-path network design problem

- Same computation time: probability of finding a solution at least as good as the target value increases from GRASP  $\rightarrow$  G+PR(F)  $\rightarrow$  G+PR(B)  $\rightarrow$  G+PR(BF)
- $P(h,t)$  = probability that variant  $h$  finds a solution as good as the target value in time no greater than  $t$ 
  - $P(\text{GRASP}, 10s) = 2\%$        $P(\text{G+PR(F)}, 10s) = 56\%$   
 $P(\text{G+PR(B)}, 10s) = 75\%$        $P(\text{G+PR(BF)}, 10s) = 84\%$



# Variants of GRASP + PR

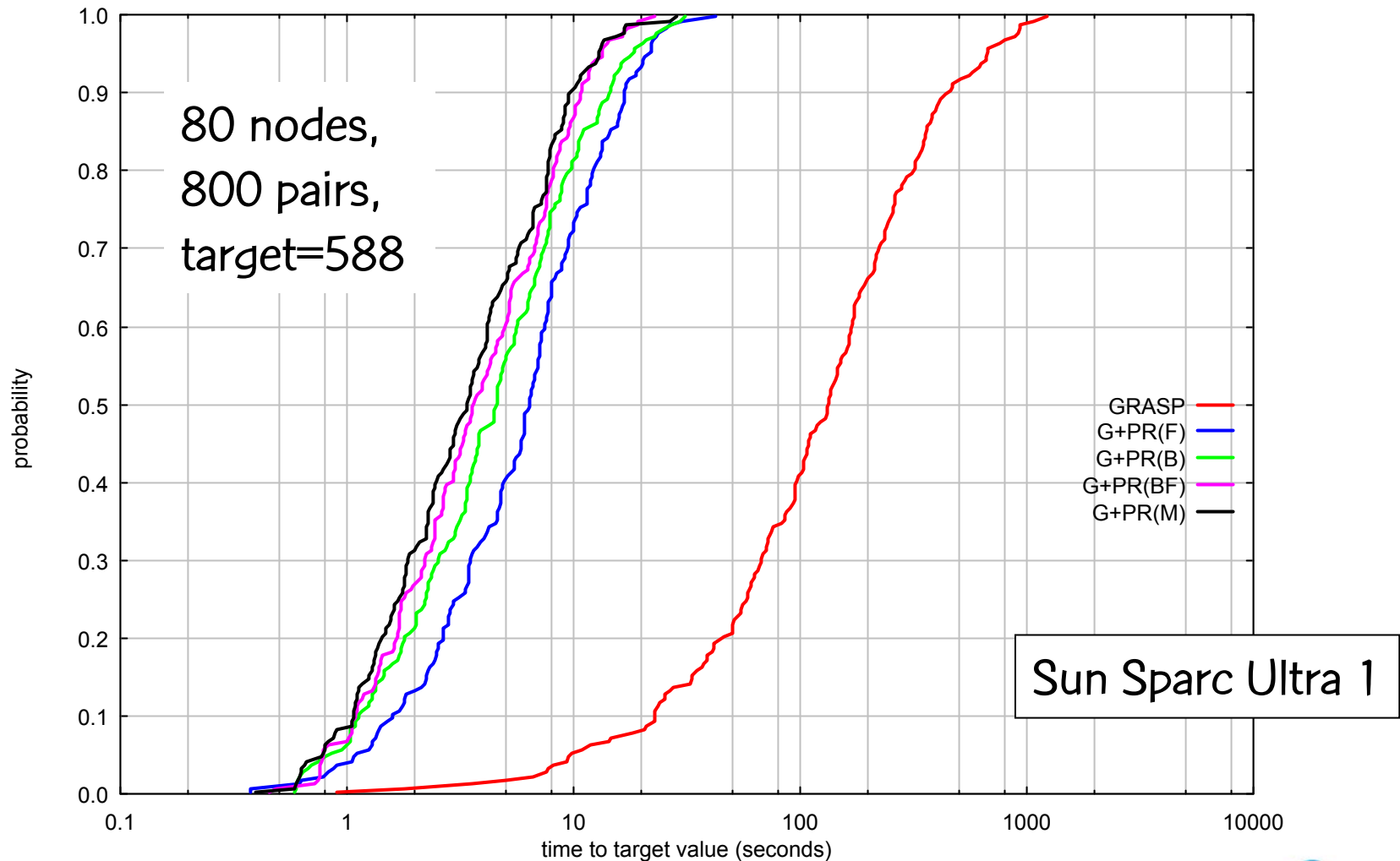
- More recently:
  - G+PR(M): mixed back and forward strategy  
*T*: elite solution    *S*: local search



- Path-relinking with local search

# 2-path network design problem

Each variant: 200 runs for one instance of 2PNDP



# 2-path network design problem

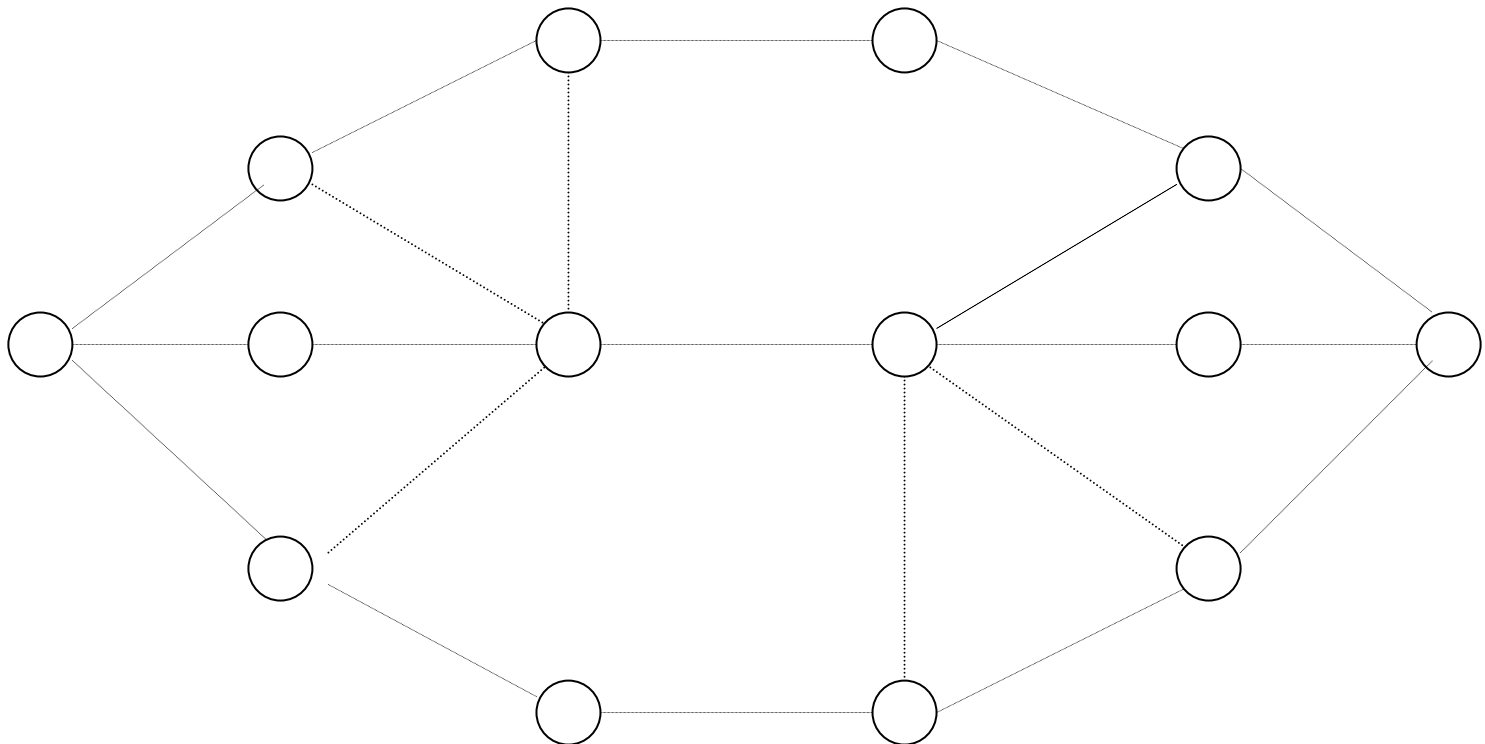
Instance	GRASP	G+PR(F)	G+PR(B)	G+PR(FB)	G+PR(M)
100-3	773	762	756	757	754
100-5	756	742	739	737	728
200-3	1564	1523	1516	1508	1509
200-5	1577	1567	1543	1529	1531
300-3	2448	2381	2339	2356	2338
300-5	2450	2364	2328	2347	2322
400-3	3388	3311	3268	3227	3257
400-5	3416	3335	3267	3270	3259
500-3	4347	4239	4187	4170	4187
500-5	4362	4263	4203	4211	4200

10 runs,  
same  
computation  
time for each  
variant,  
best solution  
found

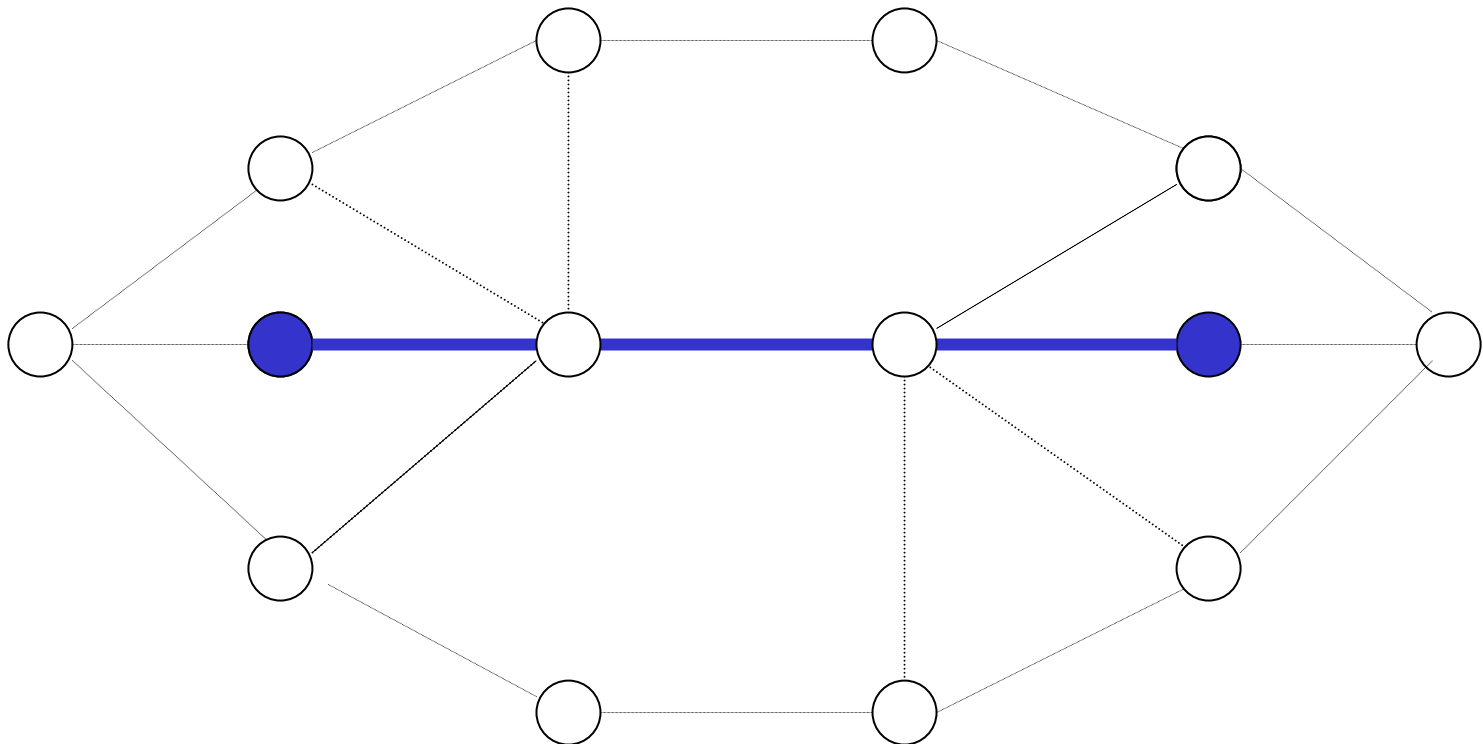
# PVC routing

- Frame relay service offers virtual private networks to customers by providing long-term private virtual circuits (PVCs) between customer endpoints on a backbone network.
- Routing is done either automatically by switch or by the network designer without any knowledge of future requests.
- Over time, these decisions cause inefficiencies in the network and occasionally offline rerouting (grooming) of the PVCs is needed:
  - integer multicommodity network flow problem: Resende & Ribeiro (2003)

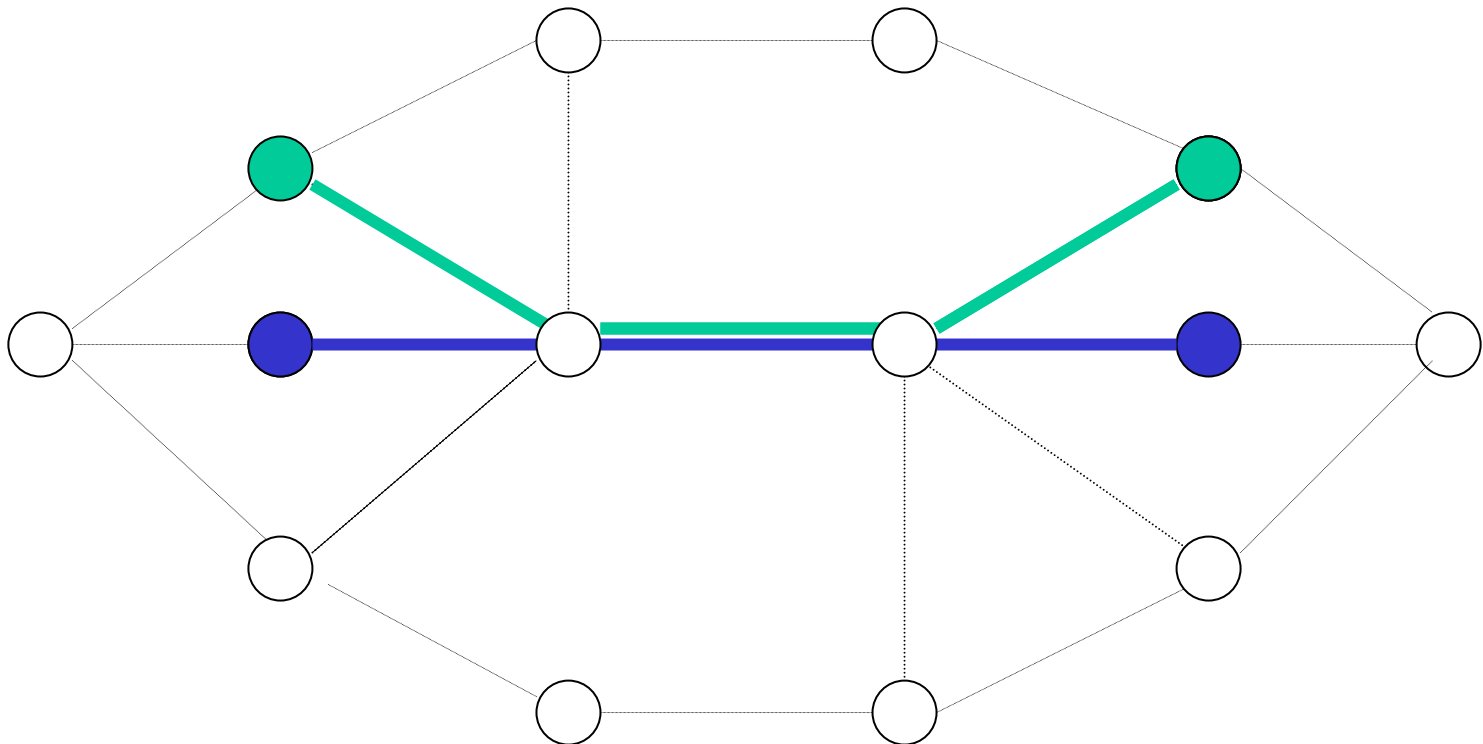
# PVC routing



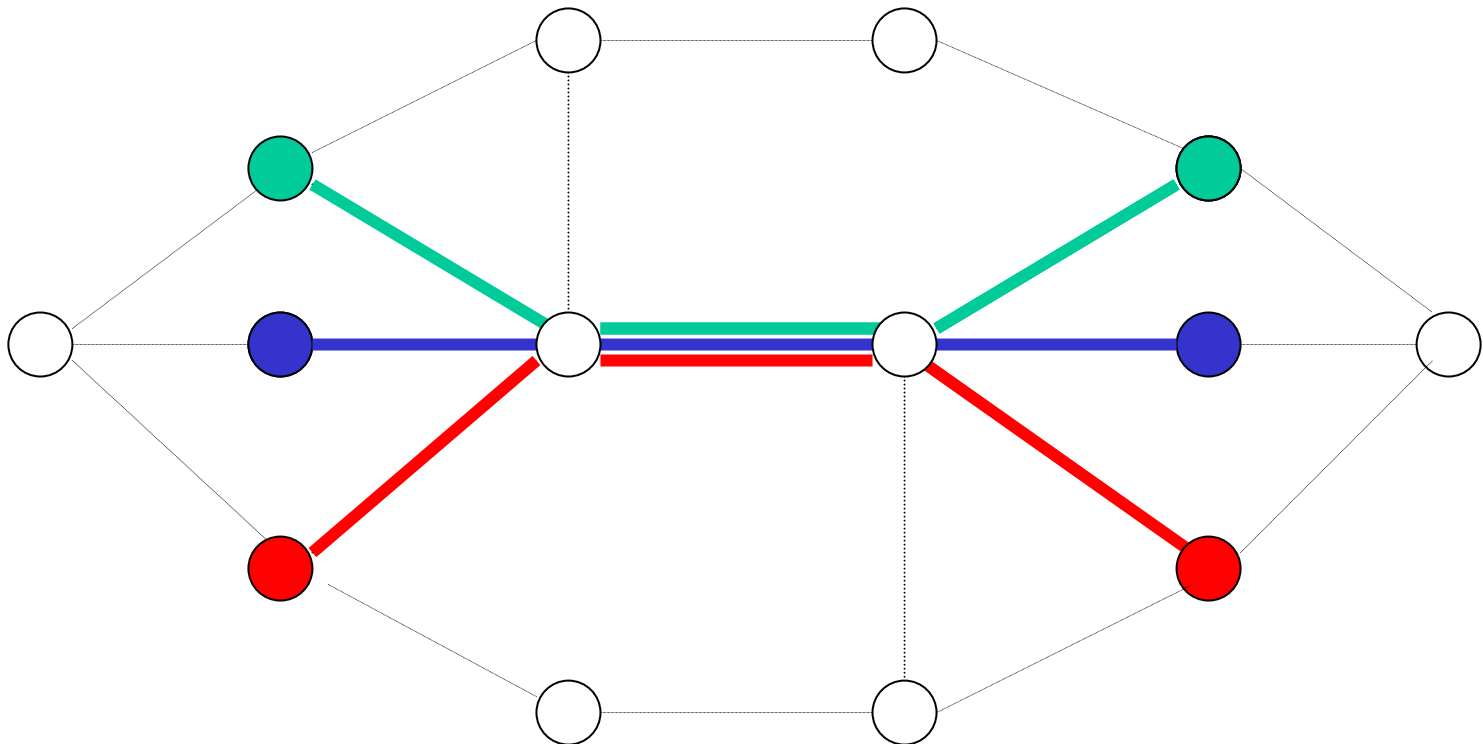
# PVC routing



# PVC routing

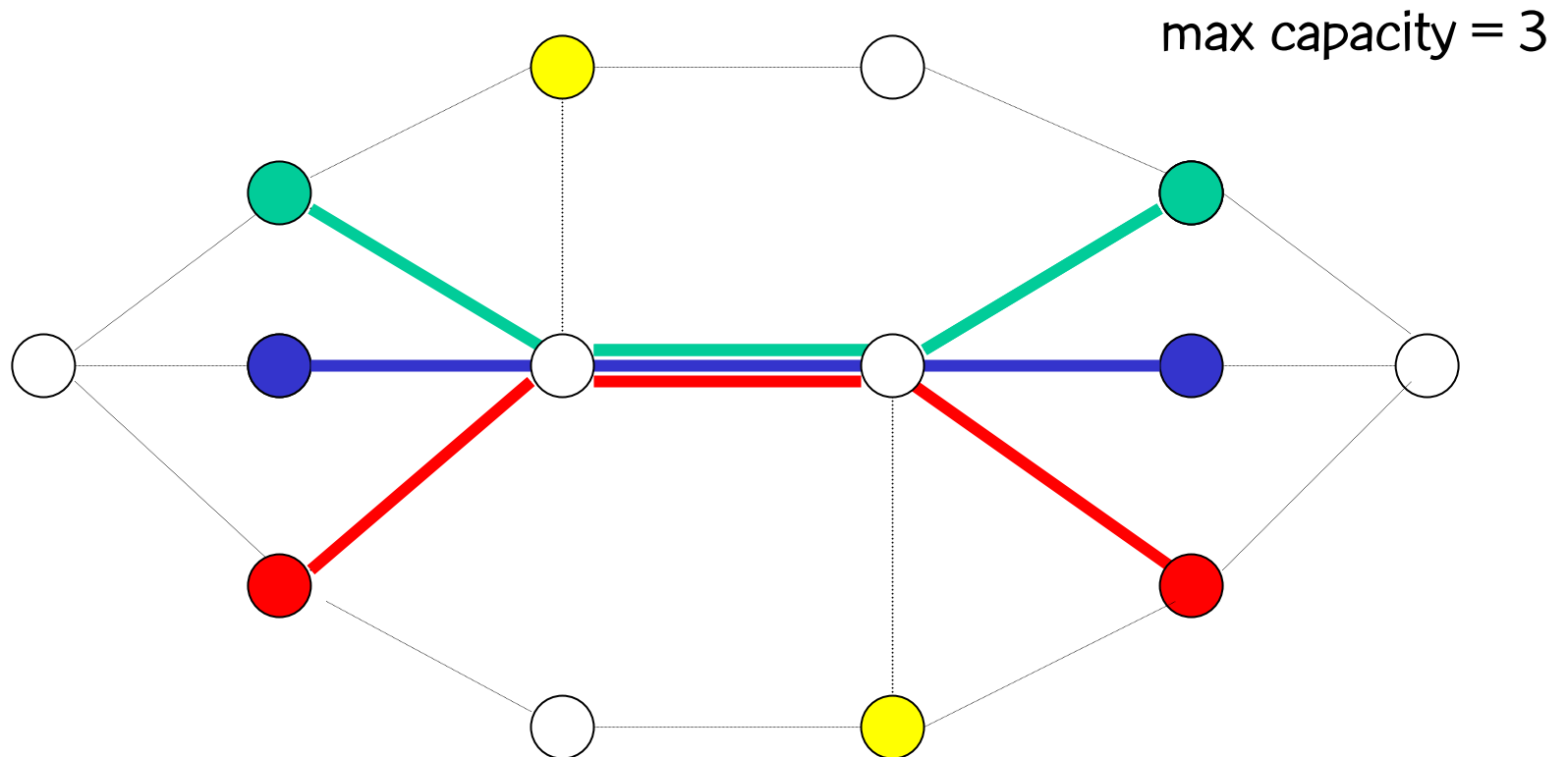


# PVC routing

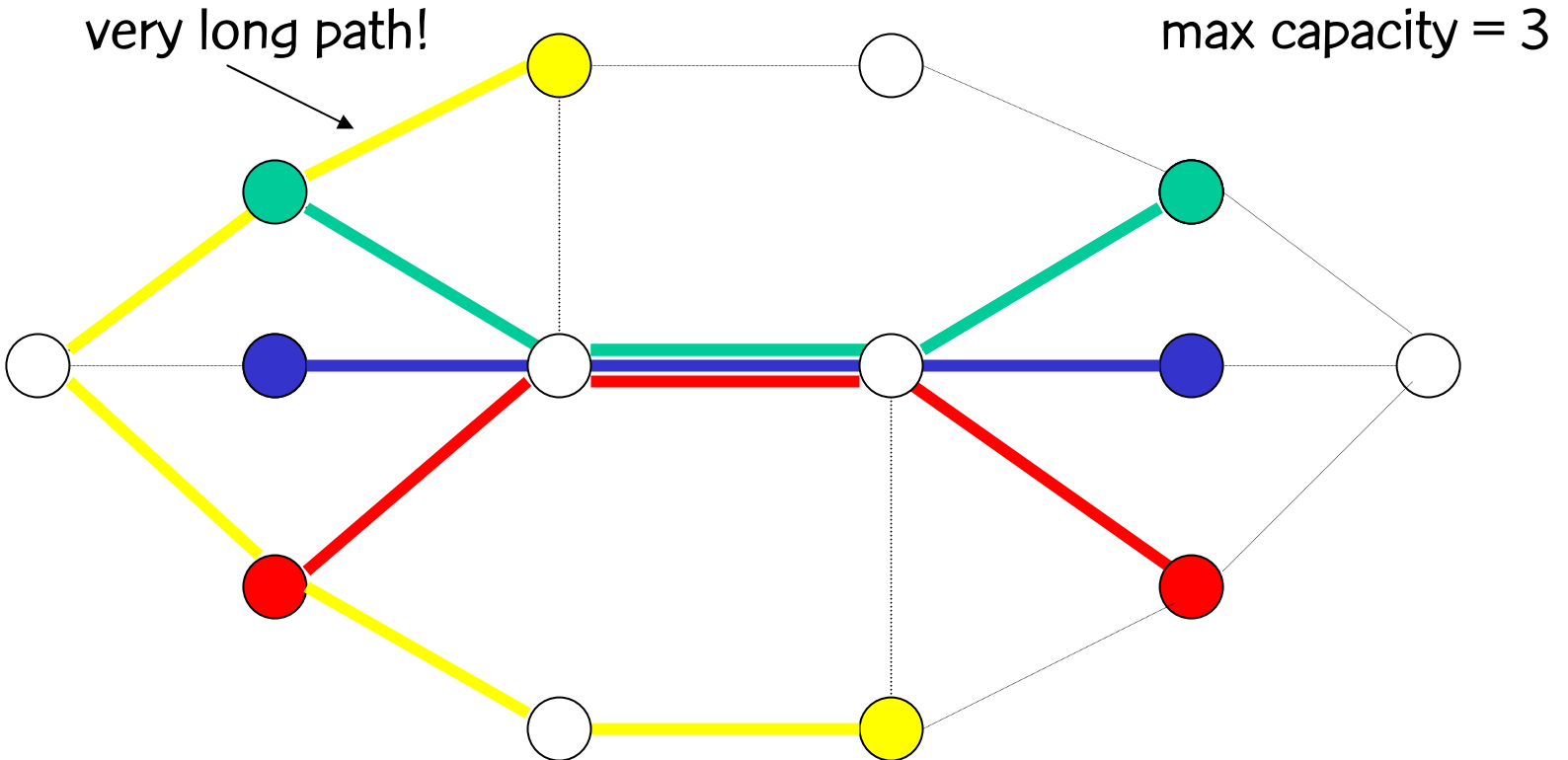




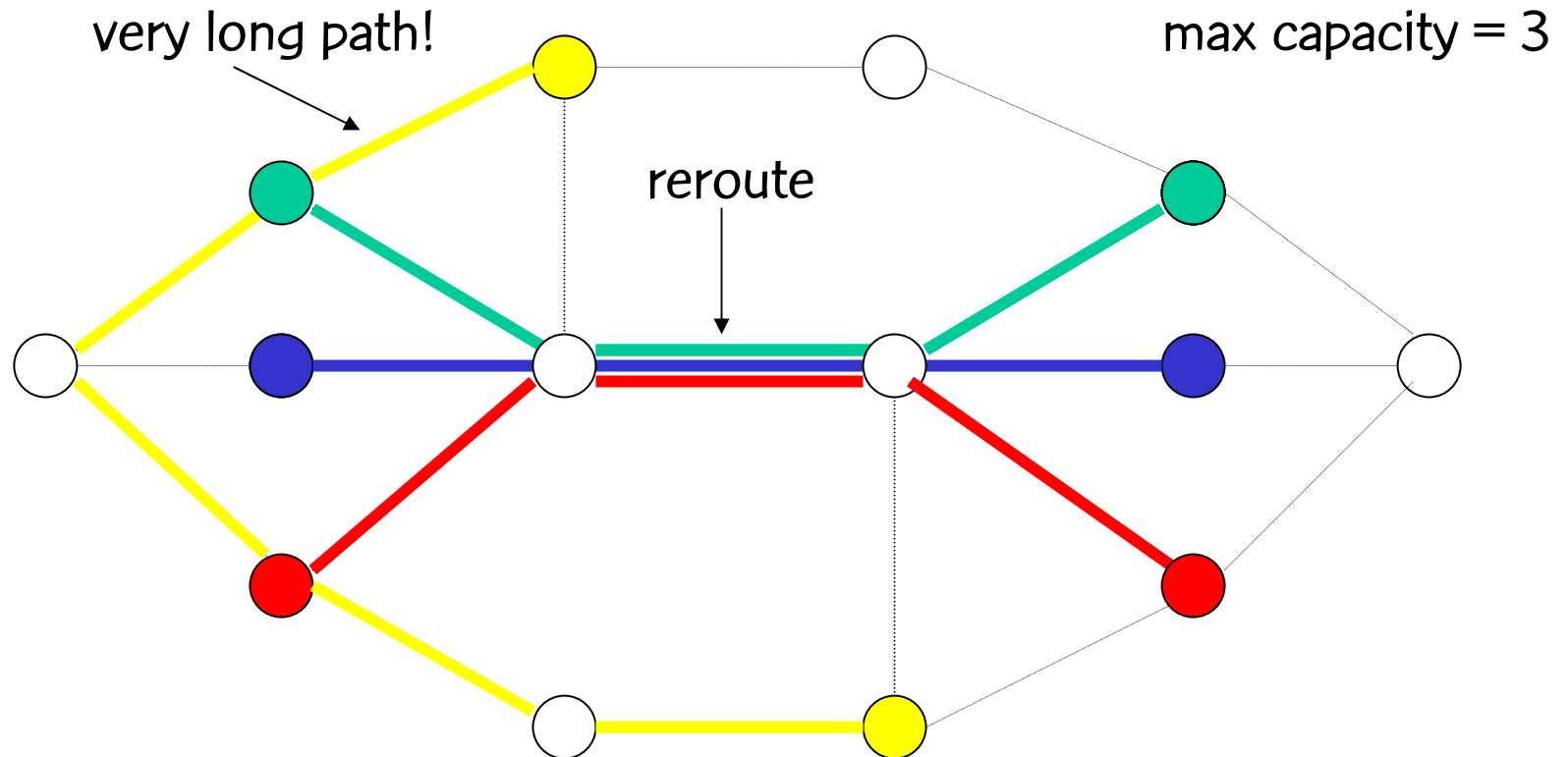
# PVC routing



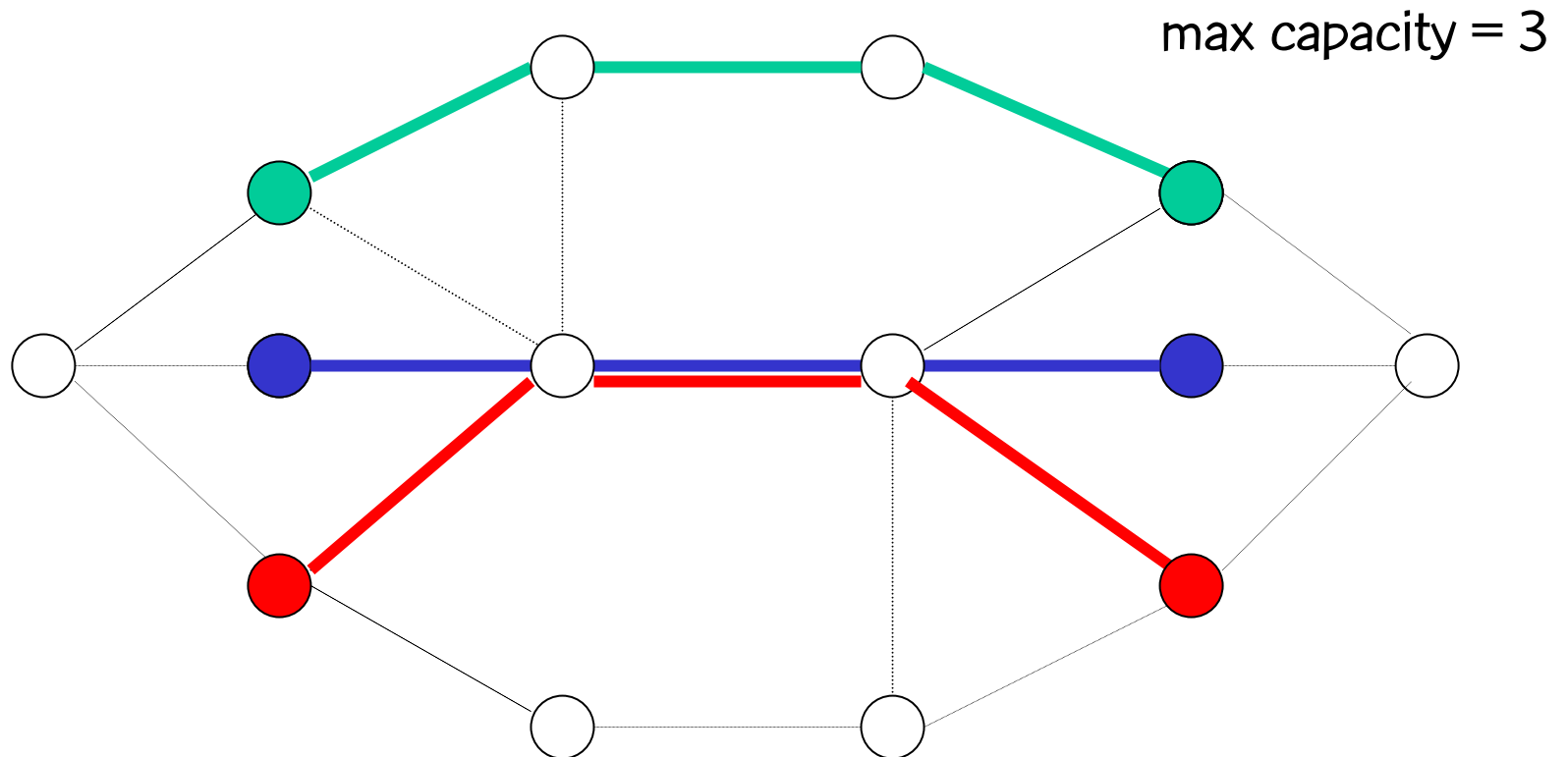
# PVC routing



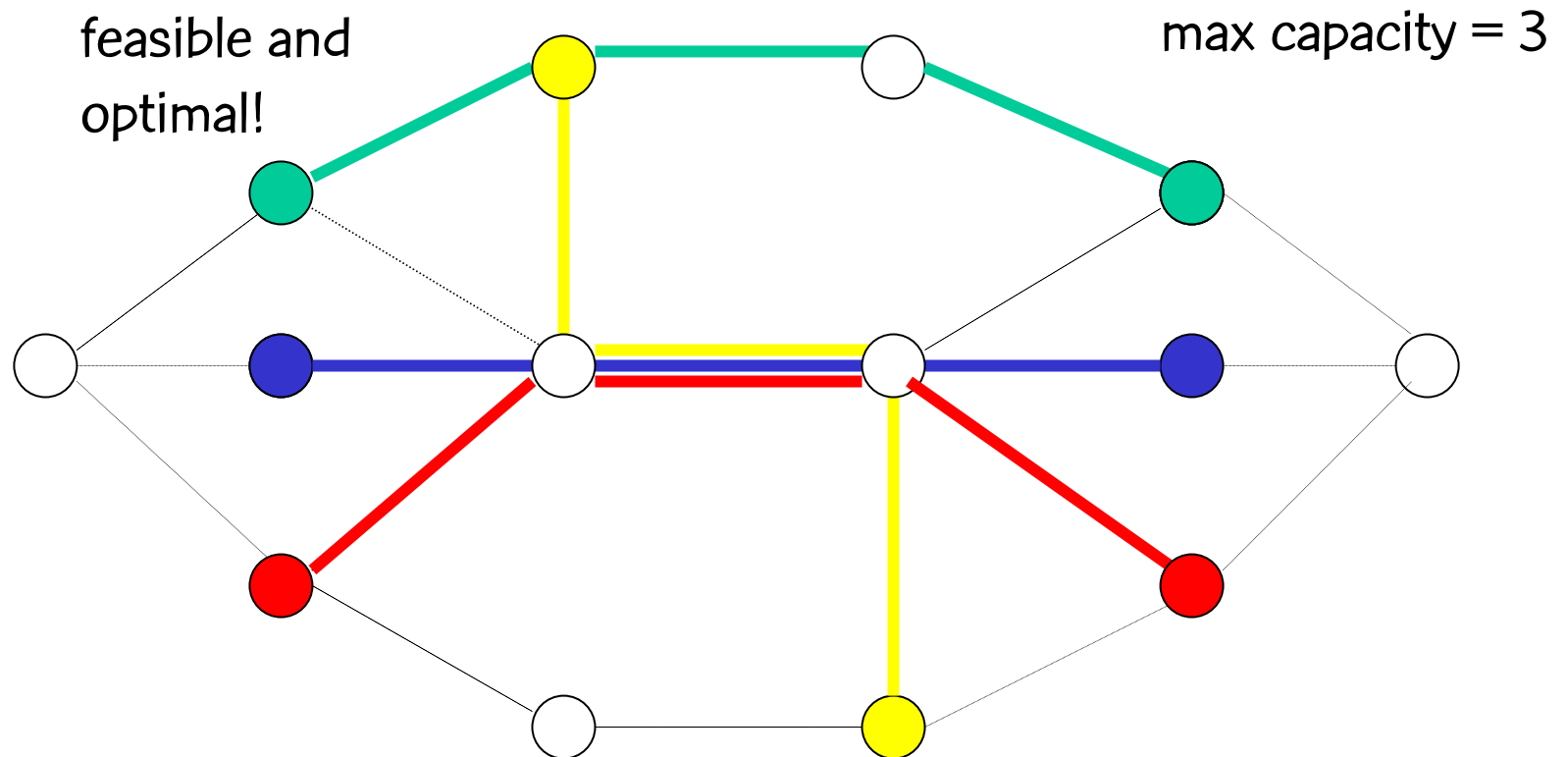
# PVC routing



# PVC routing

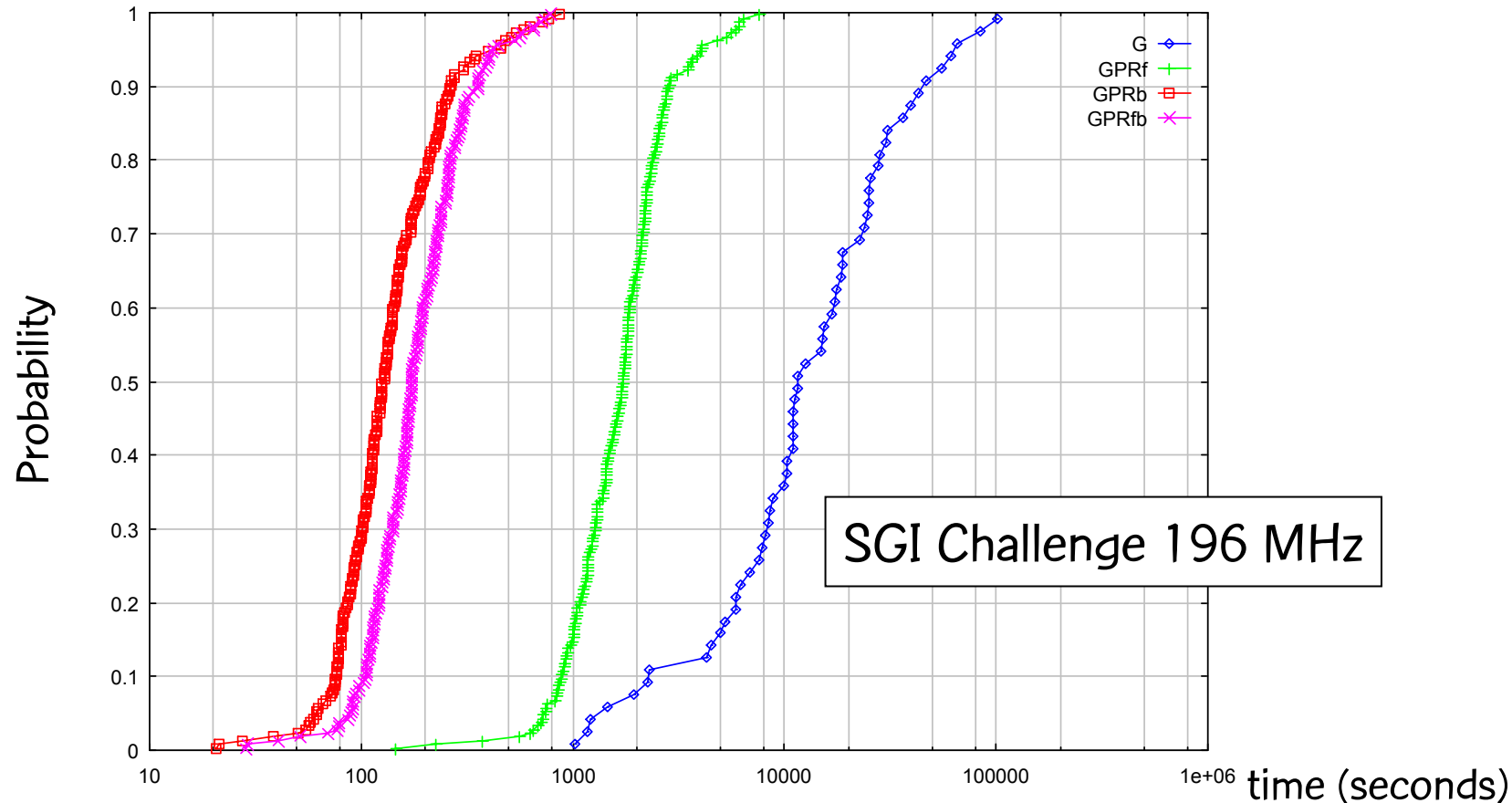


# PVC routing



# PVC routing

Each variant: 200 runs for one instance of PVC routing problem  
(60 nodes, 498 edges, 750 origin-destination pairs)



# PVC routing

10 runs	10 seconds		100 seconds	
Variant	best	average	best	average
GRASP	126603	126695	126228	126558
G+PR(F)	126301	126578	126083	126229
G+PR(B)	125960	126281	125666	125883
G+PR(BF)	125961	126307	125646	125850

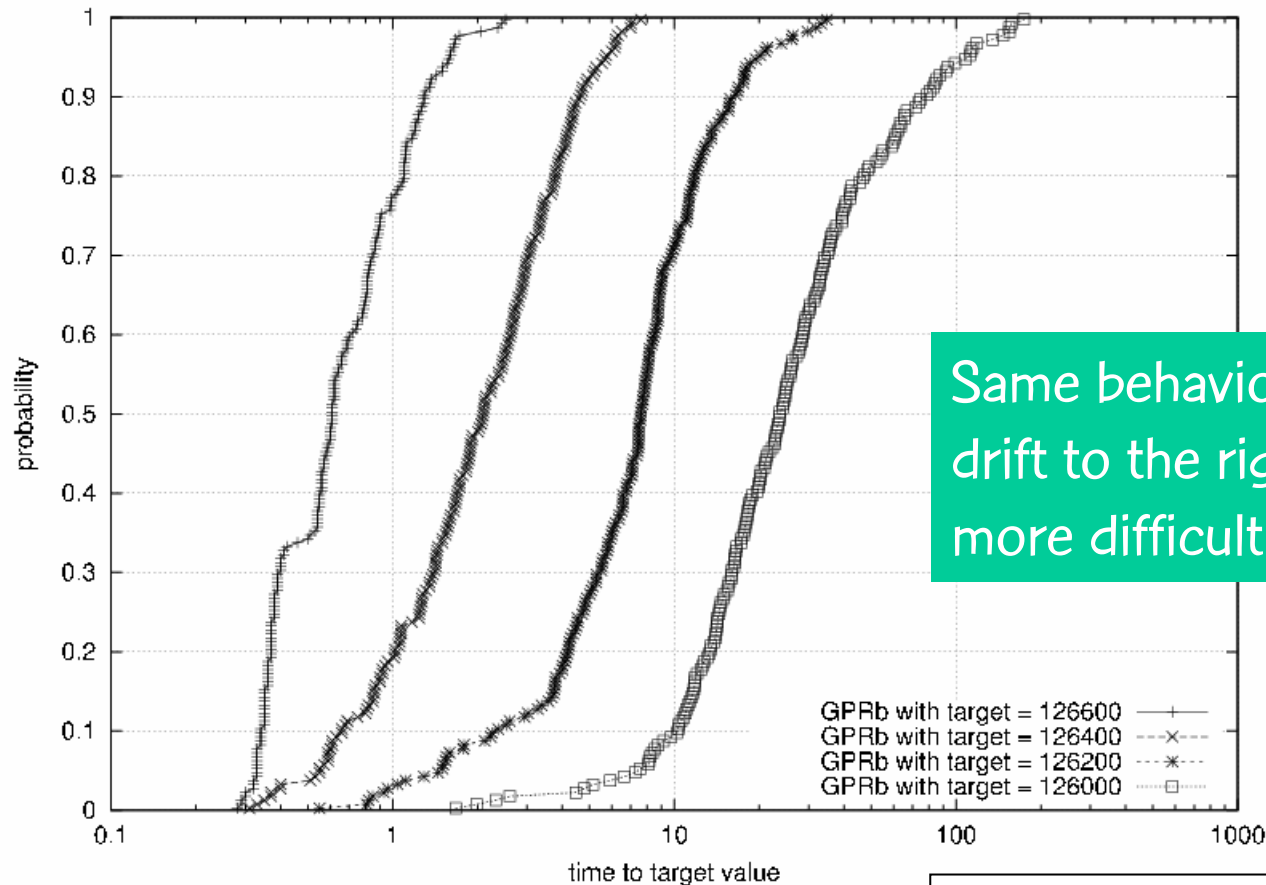
# PVC routing

10 runs	10 seconds		100 seconds	
Variant	best	average	best	average
GRASP	126603	126695	126228	126558
G+PR(F)	126301	126578	126083	126229
G+PR(B)	125960	126281	125666	125883
G+PR(BF)	125961	126307	125646	125850



# PVC routing

GRASP + PR backwards: four increasingly difficult target values



Same behavior, plots drift to the right for more difficult targets

SGI Challenge 196 MHz

# GRASP with path-relinking

Post-optimization “evolutionary” strategy:

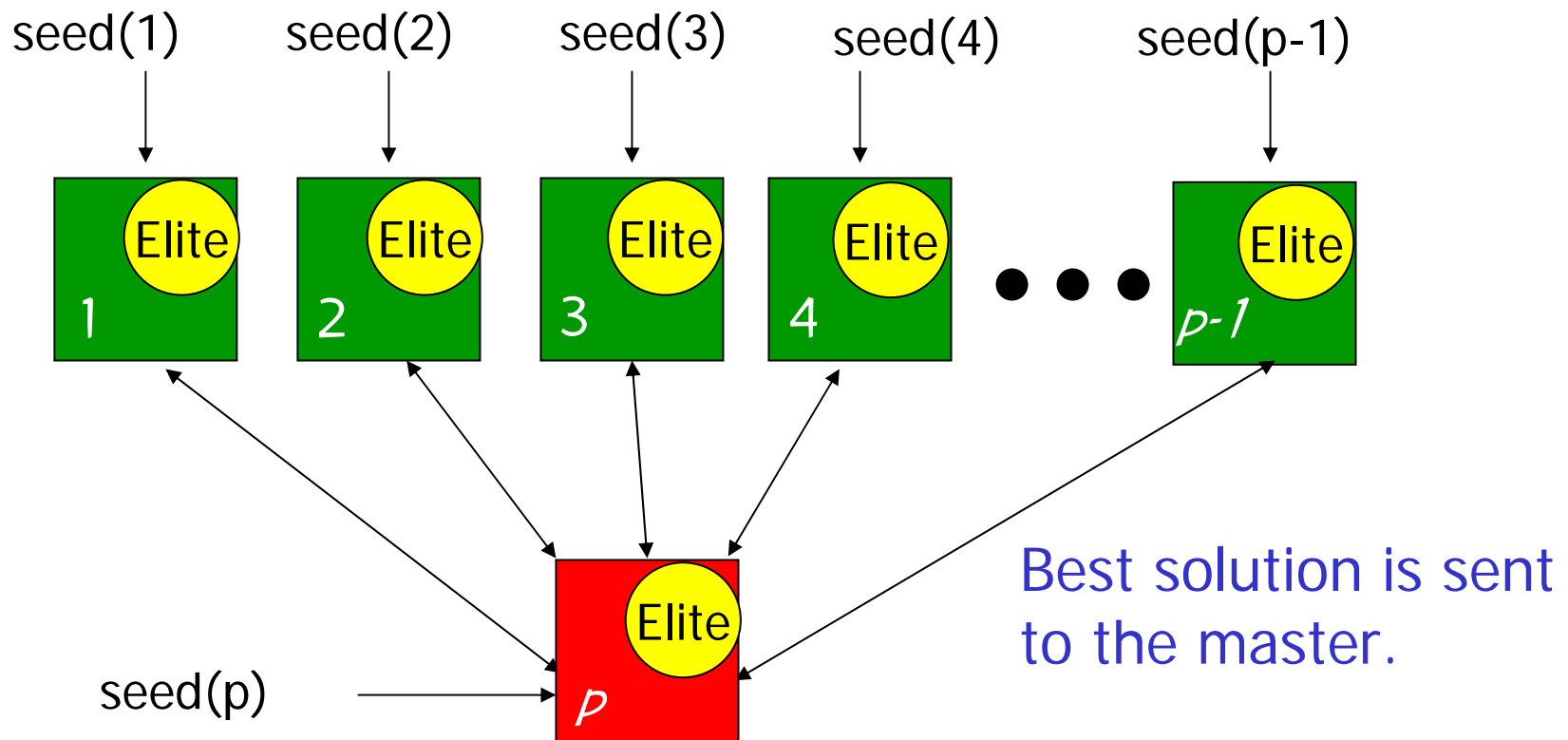
- a) **Start** with pool  $P_0$  found at end of GRASP and set  $k = 0$ .
- b) **Combine** with path-relinking all **pairs of solutions** in  $P_k$ .
- c) Solutions obtained by combining **solutions** in  $P_k$  are **added to a new pool**  $P_{k+1}$  following same constraints for updates as before.
- d) If **best solution** of  $P_{k+1}$  **is better** than best solution of  $P_k$ , then set  $k = k + 1$ , and go back to step (b).

Successfully used by Ribeiro, Uchoa, & Werneck (2002)  
(Steiner Problem in Graphs) and Resende & Werneck  
(2002-3) (p-median & facility location)

# Parallel independent implementation

- Parallelism in metaheuristics: **robustness**  
Duni-Eksioglu, Pardalos, and Resende (2002)
- Multiple-walk independent-thread strategy:
  - $p$  processors available
  - Iterations evenly distributed over  $p$  processors
  - Each processor keeps a copy of data and algorithms.
  - One processor acts as the **master** handling seeds, data, and iteration counter, **besides performing GRASP iterations.**
  - Each processor performs  $\text{Max\_Iterations}/p$  iterations.

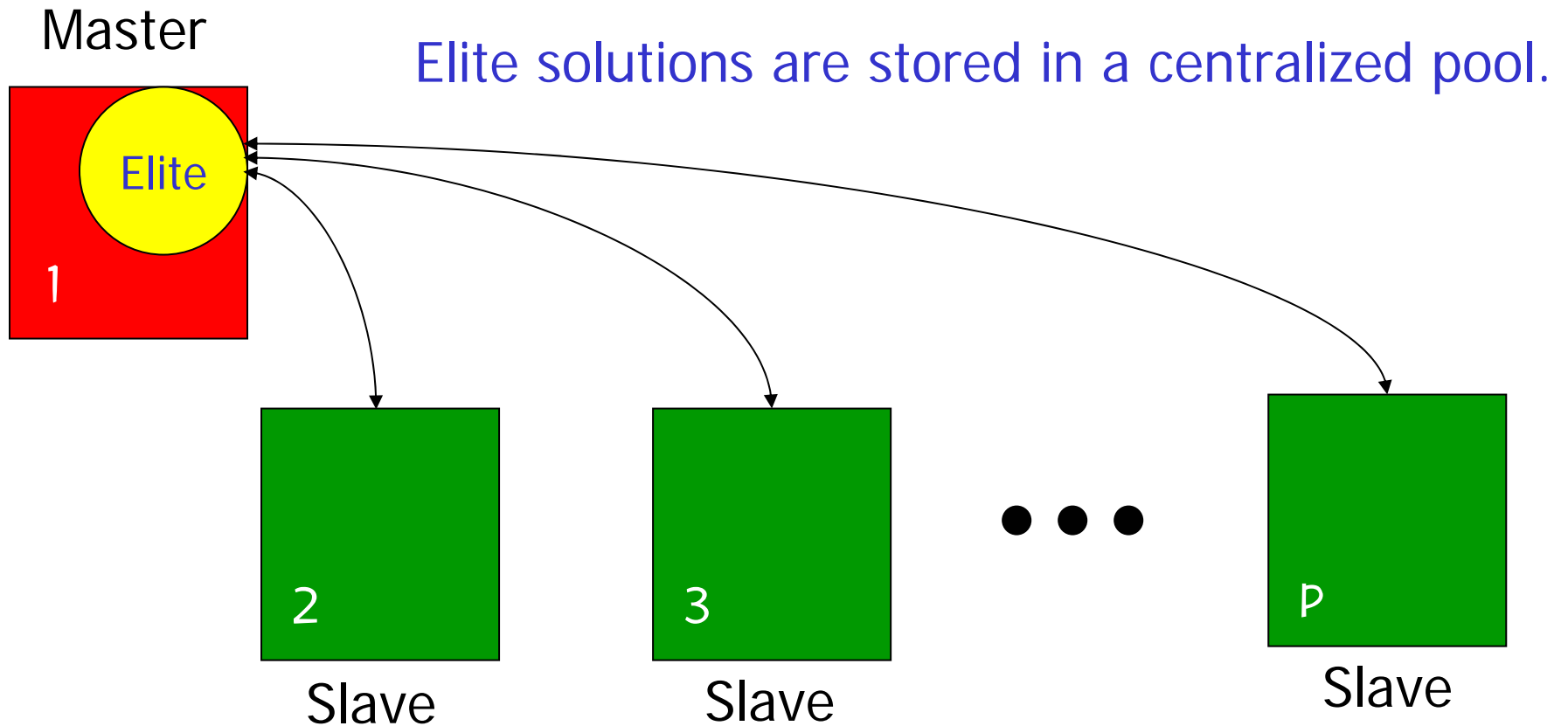
# Parallel independent implementation



# Parallel cooperative implementation

- Multiple-walk cooperative-thread strategy:
  - $p$  processors available
  - Iterations evenly distributed over  $p-1$  processors
  - Each processor has a copy of data and algorithms.
  - One processor acts as the **master** handling seeds, data, and iteration counter and **handles the pool of elite solutions**, but does not **perform GRASP iterations**.
  - Each processor performs  $\text{Max\_Iterations}/(p-1)$  iterations.

# Parallel cooperative implementation

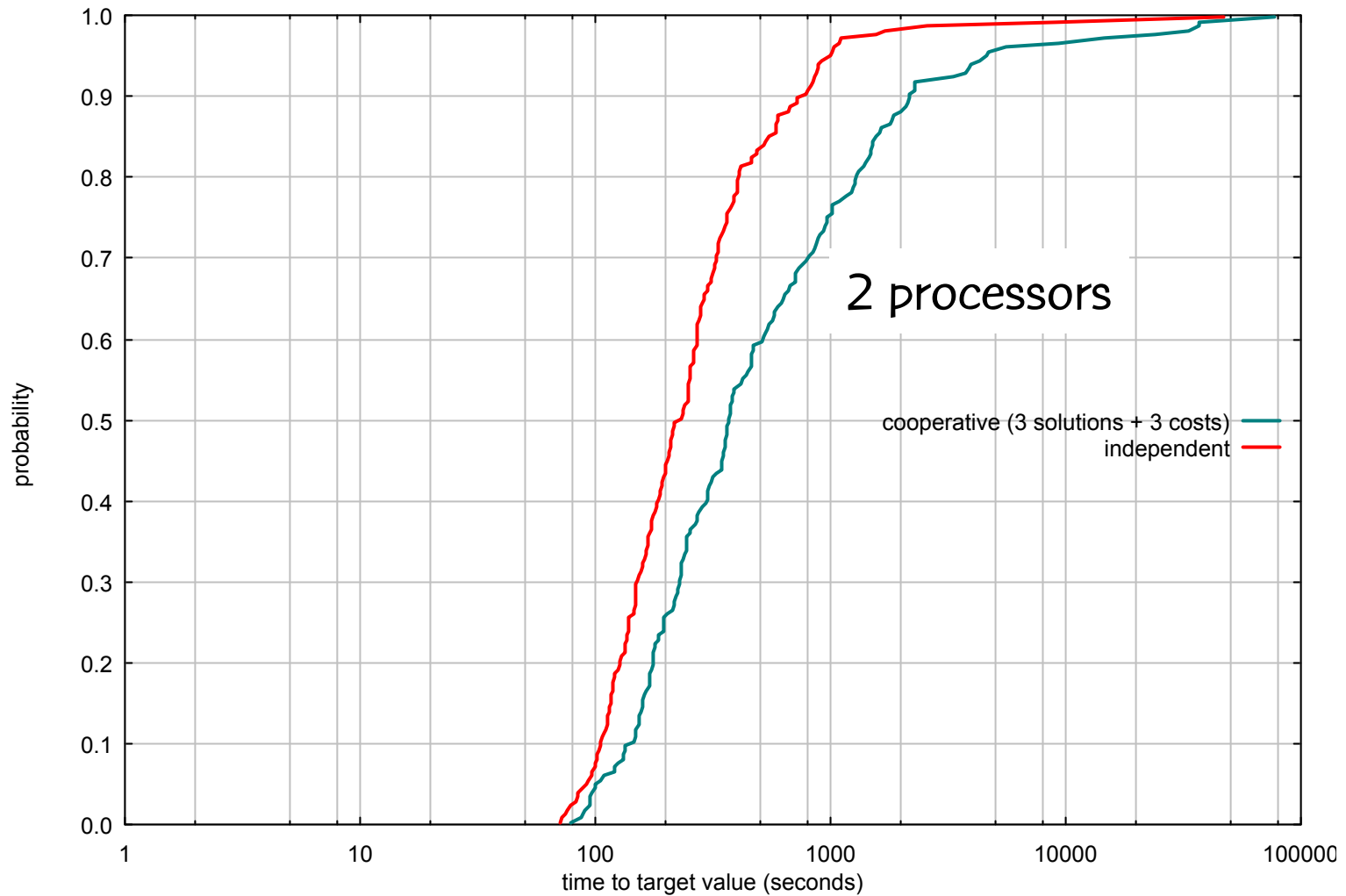


# Parallel environment at PUC-Rio

- Linux cluster with 32 Pentium IV 1.7 GHz processors with 256 Mbytes of RAM each
- Extreme Networks switch with 48 10/100 Mbits/s ports and two 1 Gbits/s ports

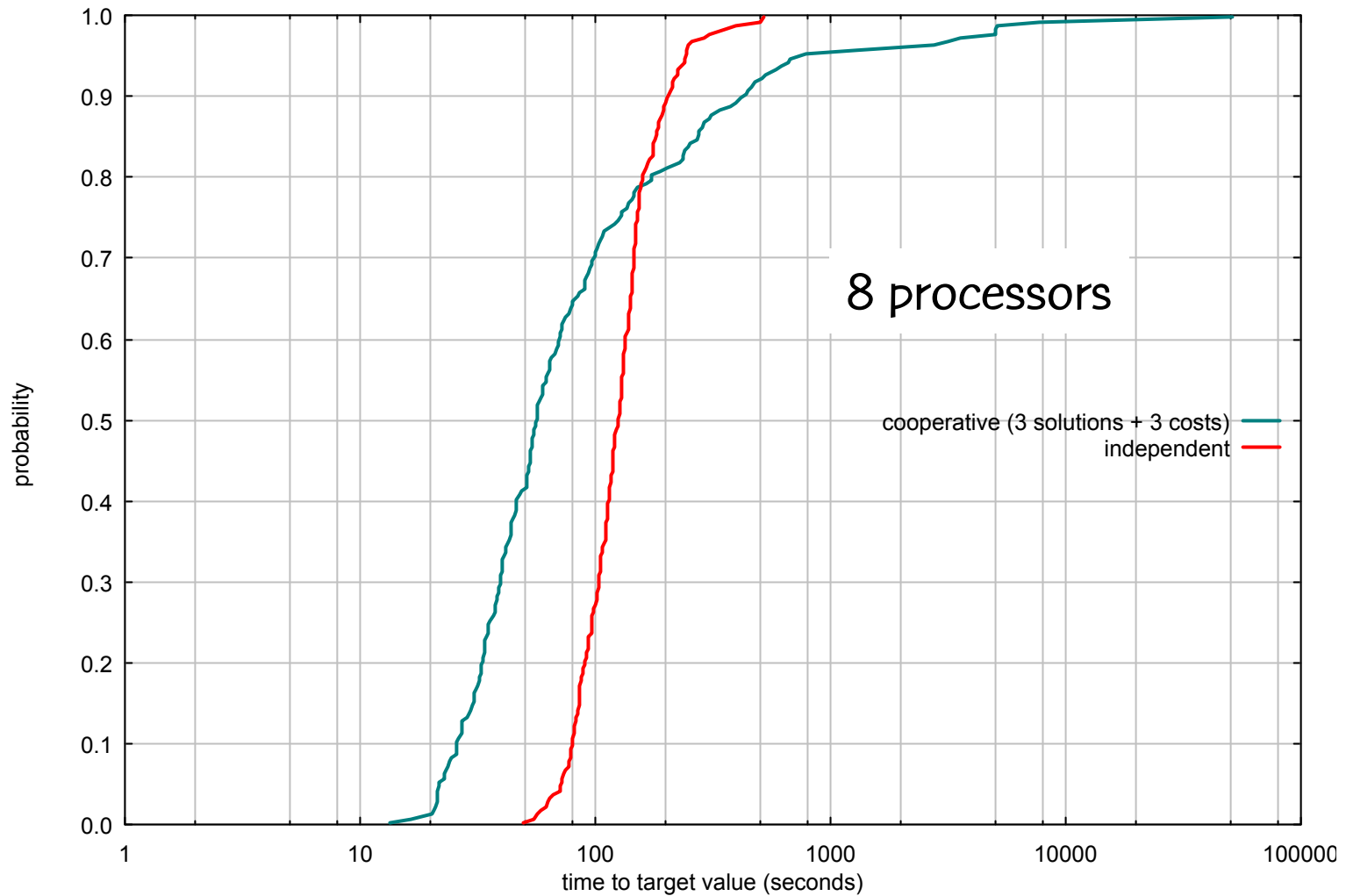


# Parallel environment

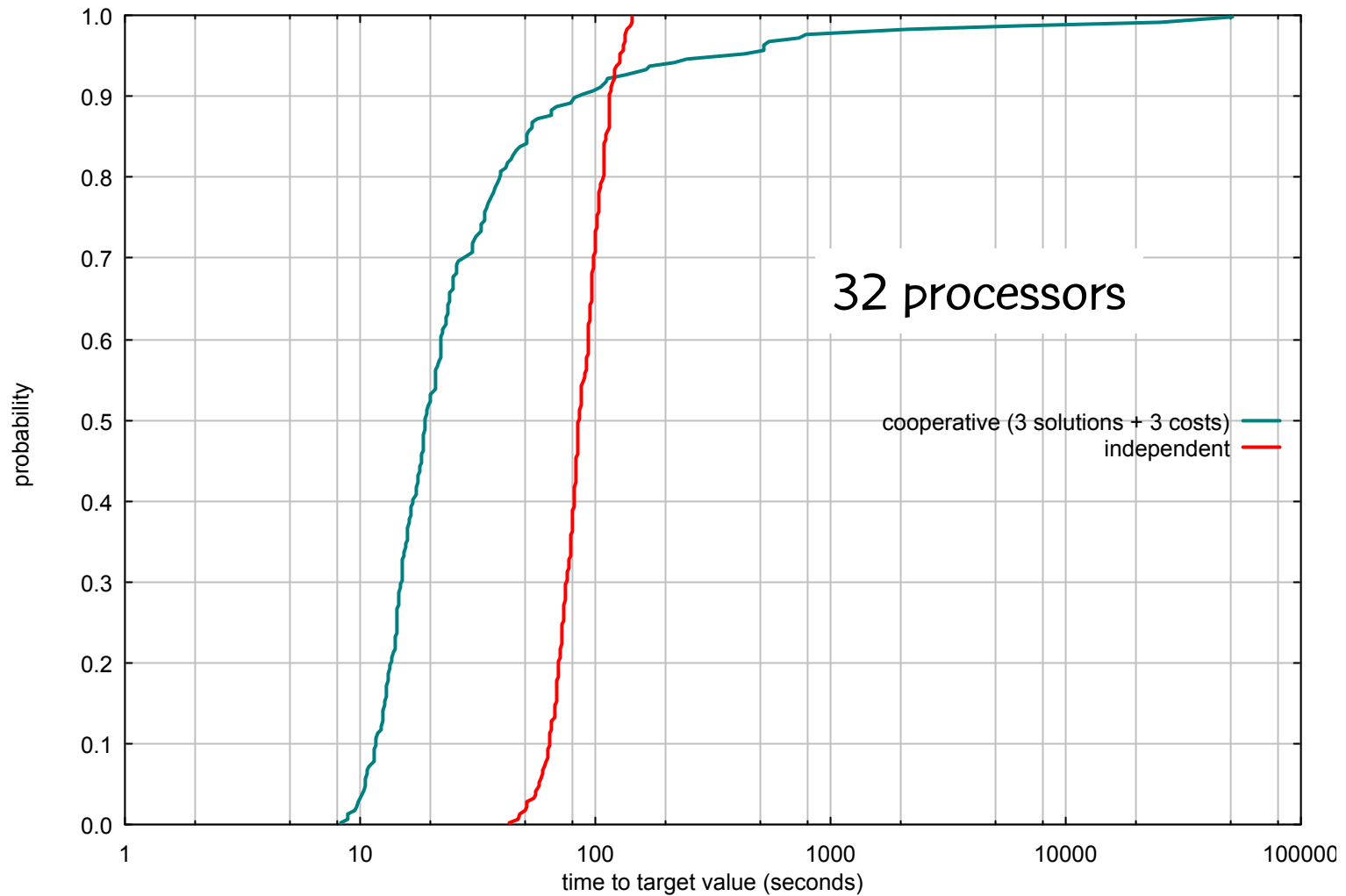




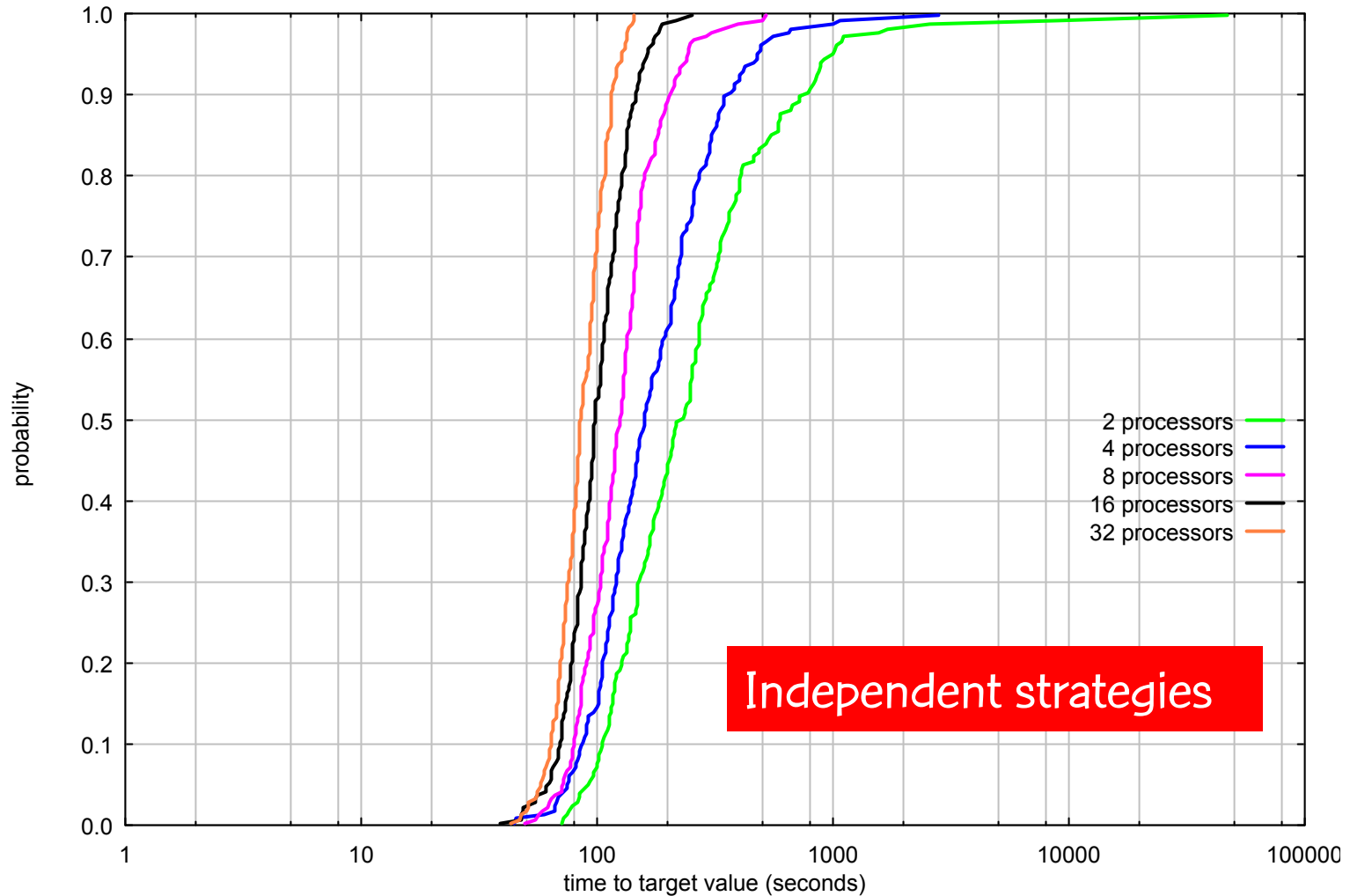
# Parallel environment



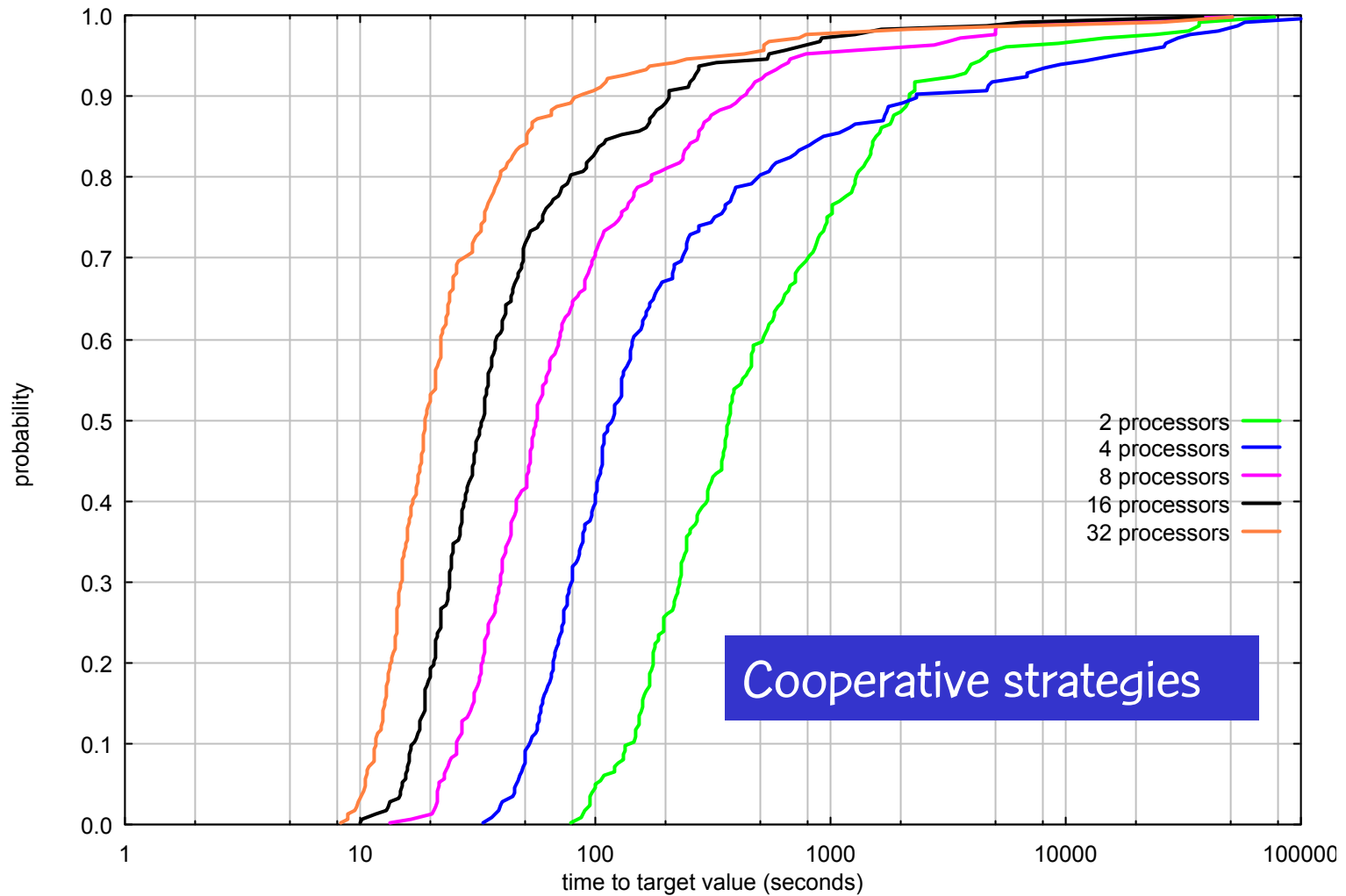
# Parallel environment



# Parallel environment



# Parallel environment



# Concluding remarks of Part 1

Path-relinking adds memory and intensification mechanisms to GRASP, systematically contributing to improve solution quality:

- better solutions in smaller times
- some implementation strategies appear to be more effective than others.
- mixed path-relinking strategy is very promising
- backward relinking is usually more effective than forward
- bidirectional relinking does not necessarily pay off the additional computation time

# Concluding remarks of Part 1

## Cooperative parallel strategies based on path-relinking:

- Path-relinking offers a nice strategy to introduce memory and cooperation in parallel implementations.
- Cooperative strategy performs better due to smaller number of iterations and to inter-processor cooperation.
- Linear speedups with the parallel implementation.
- Robustness: cooperative strategy is faster and better.
- Parallel systems are not easily scalable, parallel strategies require careful implementations.

Application 1:

Modem pool location for  
dial-up ISP access

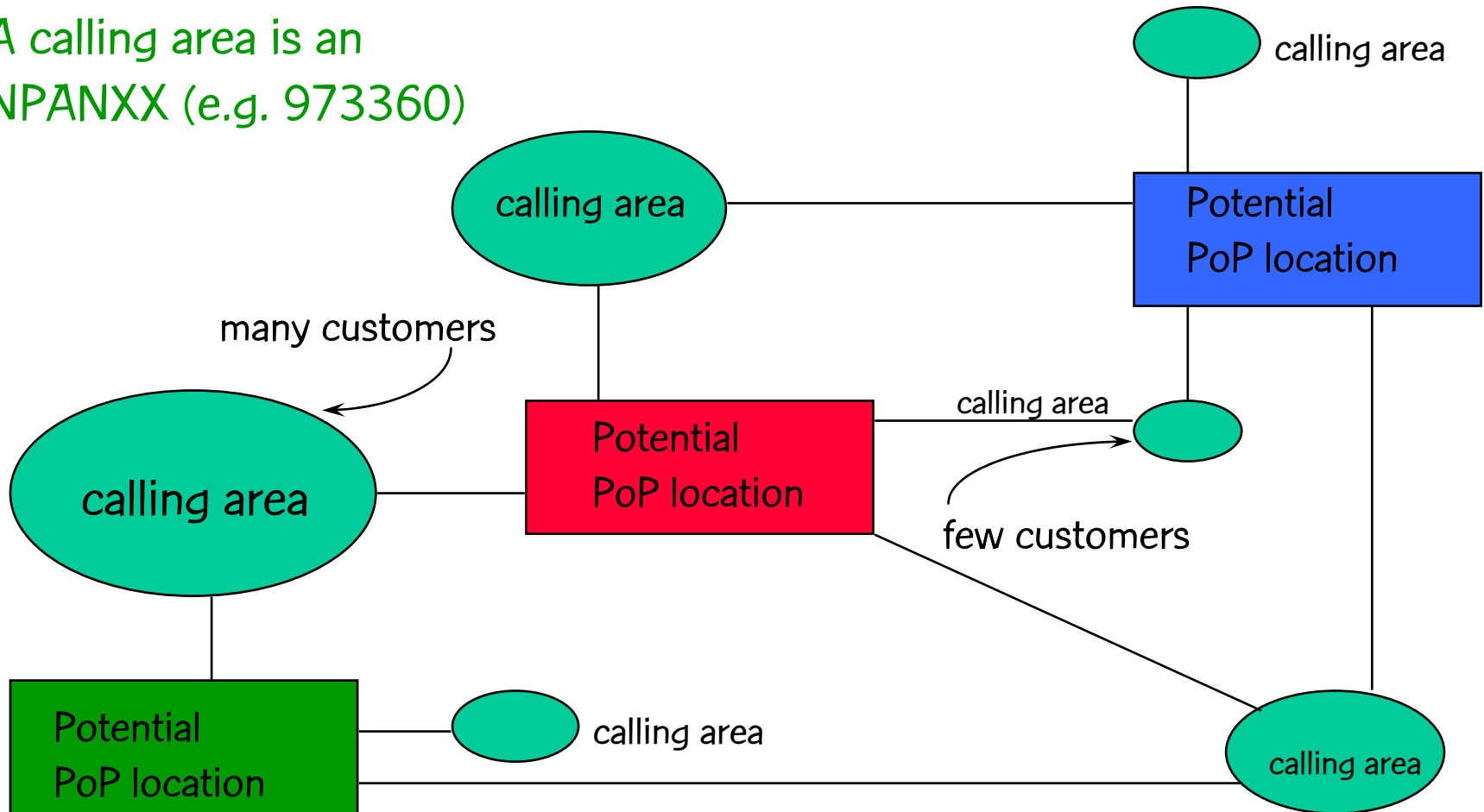
# Modem pool location for dial-up ISP access

- user dials up to a modem to access an internet service provider
- modem pools are located at PoPs (points of presence)
- users prefer making free local calls to access internet service



# ISP access

A calling area is an  
NPANXX (e.g. 973360)



# Location problem

- maximize number of customers that can make free local calls to a PoP
- where to locate PoPs
  - fixed number of PoPs
  - choose from set of potential PoP locations

# Typical size

- ~ 60,000 potential PoP locations
- ~ 50,000 calling areas (NPANXX)
- ~ 120 million residential lines
- Initially, + 255 PoPs had to be located
  - GRASP was used for initial setup in 1996
  - GRASP has been used since then for expansion

# AT&T Worldnet

- Worldnet: AT&T's Internet Service Provider
- Dial-up: hundreds of *points of presence* (PoPs)
  - Telephone numbers customers must call when making an Internet connection.
- Current footprint:
  - 1305 PoPs;
  - 77.66% of the telephone numbers in the U.S. can make local calls to Worldnet.

# Worldnet

- When is a call local?
  - Not simply “within same area code”.
  - Telephone system divided into *exchanges*.
    - Area code + first three digits (973360, for example).
- Each PoP has a *coordinate*.
- We know which exchanges can make local calls to each coordinate (the coverage).
  - Just a big table;
  - 69,534 exchanges covered by current footprint.

# Footprint Optimization

- In general: more PoPs, better coverage.
- For a fixed coverage, we don't want more PoPs than necessary.
- Not all PoPs are the same:
  - Each has an associated **network cost**:
    - Hourly rate paid by Worldnet to network company.
    - Between \$0.04 and \$0.14 in the continental US.
    - Up to \$0.42 in Hawaii and Alaska.
  - No setup cost.
- Goal: keep only cheaper PoPs, preserve coverage.

# Footprint Optimization

- Simple improvement:
  - Some coordinates have more than one PoP;
  - 1035 unique coordinates (out of 1305);
  - Keep only the cheapest PoP in each coordinate.

# Footprint Optimization

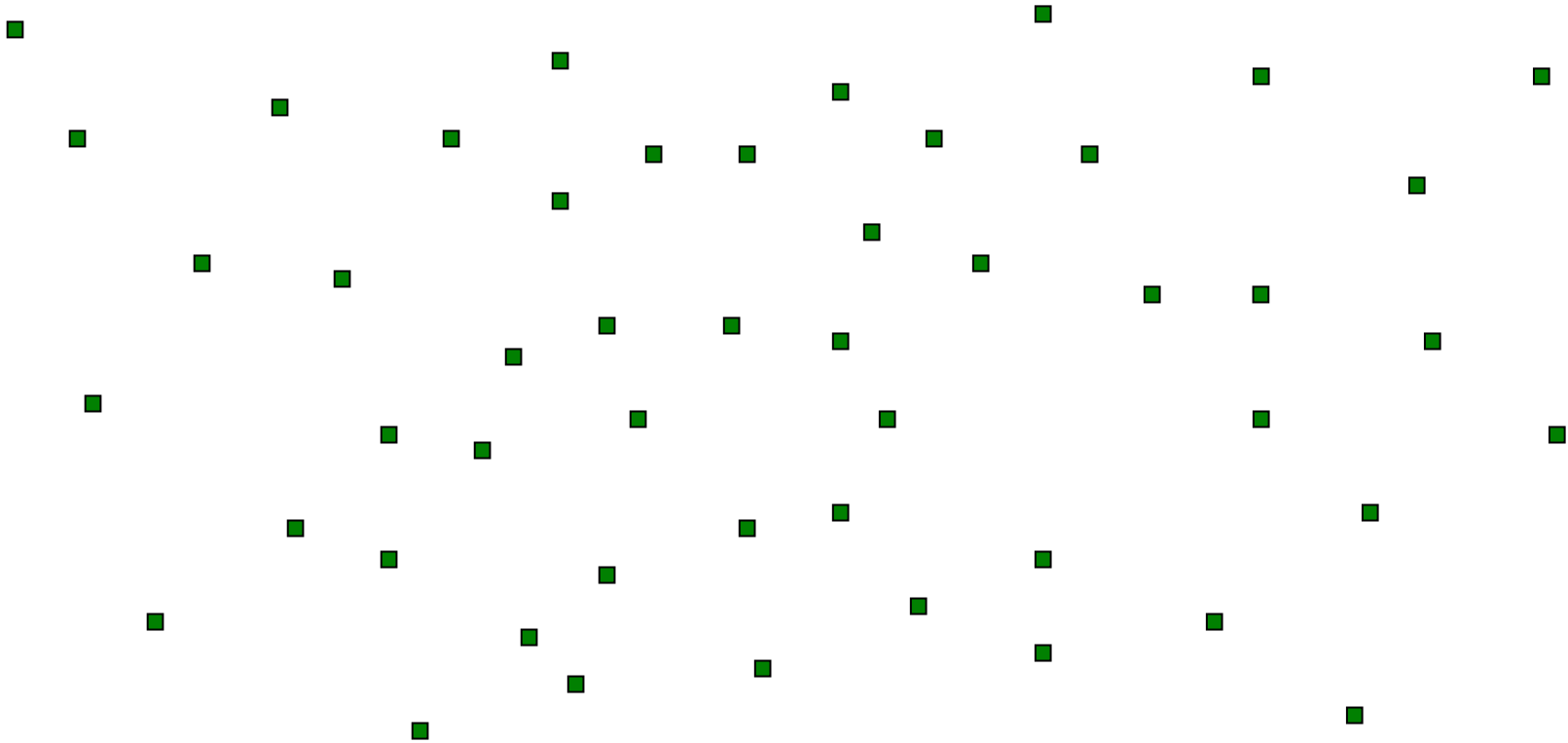
- Further improvement:
  - 335 additional coordinates could be eliminated:
    - Only 700 PoPs left;
    - New footprint covers all exchanges currently covered;
    - No exchange has to make a more expensive call.
- How did we do it?
  - We solved this as the  $p$ -median problem.



# The $p$ -median Problem

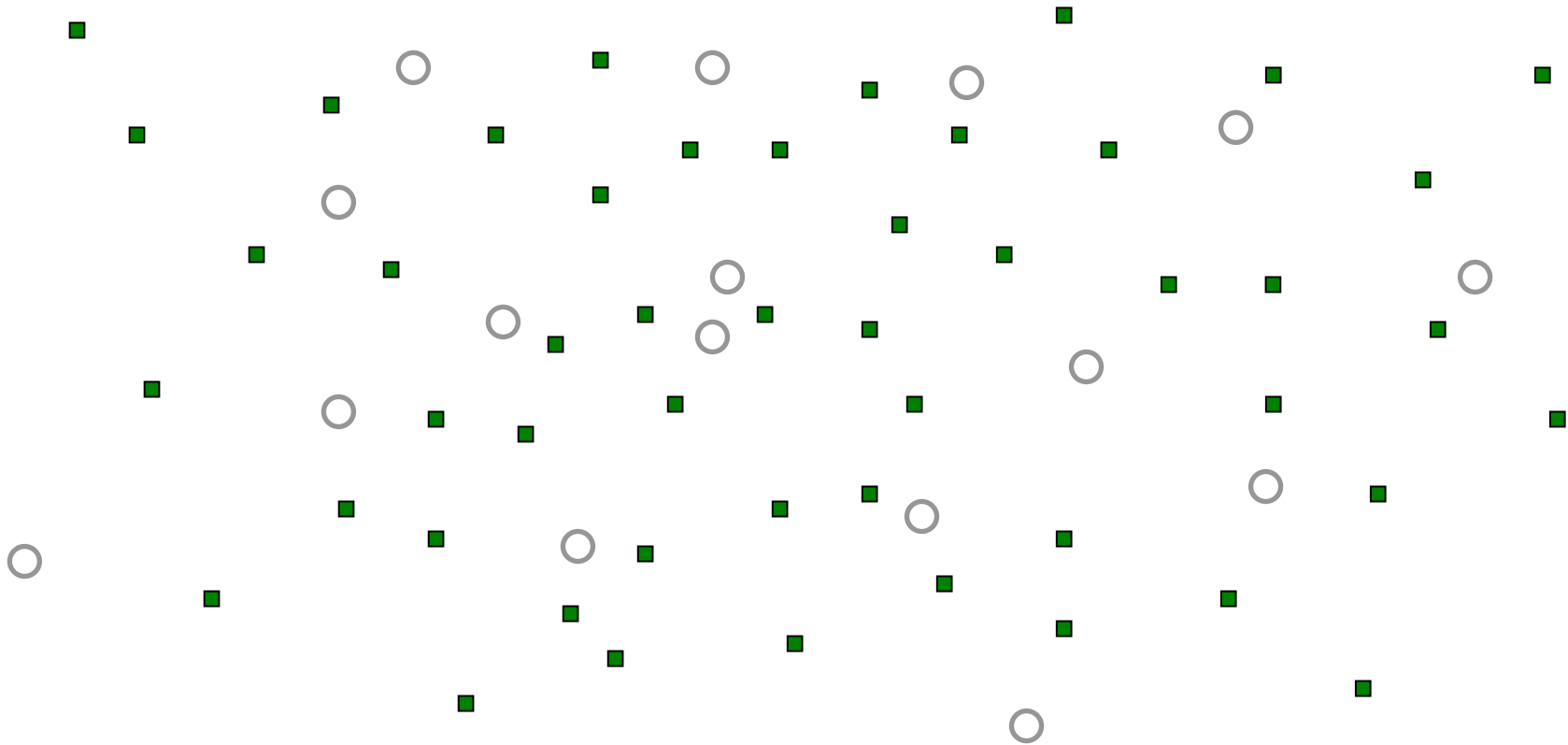
- Input:
  - a set  $C$  of  $n$  *customers* (or *users*)
  - a set  $F$  of  $m$  *potential facilities*
  - a distance function ( $d: C \times F \rightarrow \Re$ )
  - the number of facilities  $p$  to open ( $0 < p < m$ )
- Output:
  - a set  $S \subseteq F$  with  $p$  open facilities
- Goal:
  - minimize the sum of the distances from each user to the closest open facility

# Example (p-median)



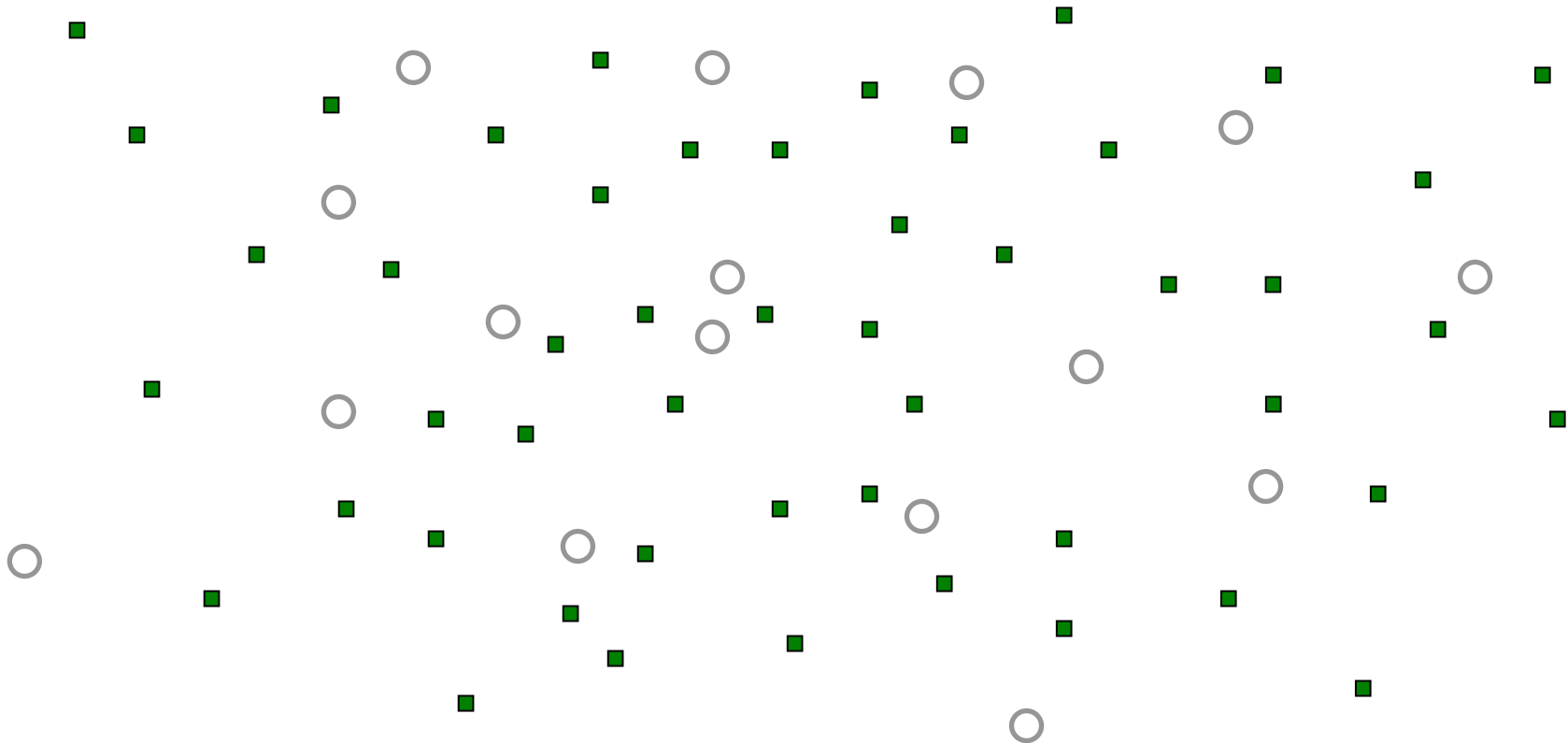
50 customers

# Example (p-median)



16 potential facilities

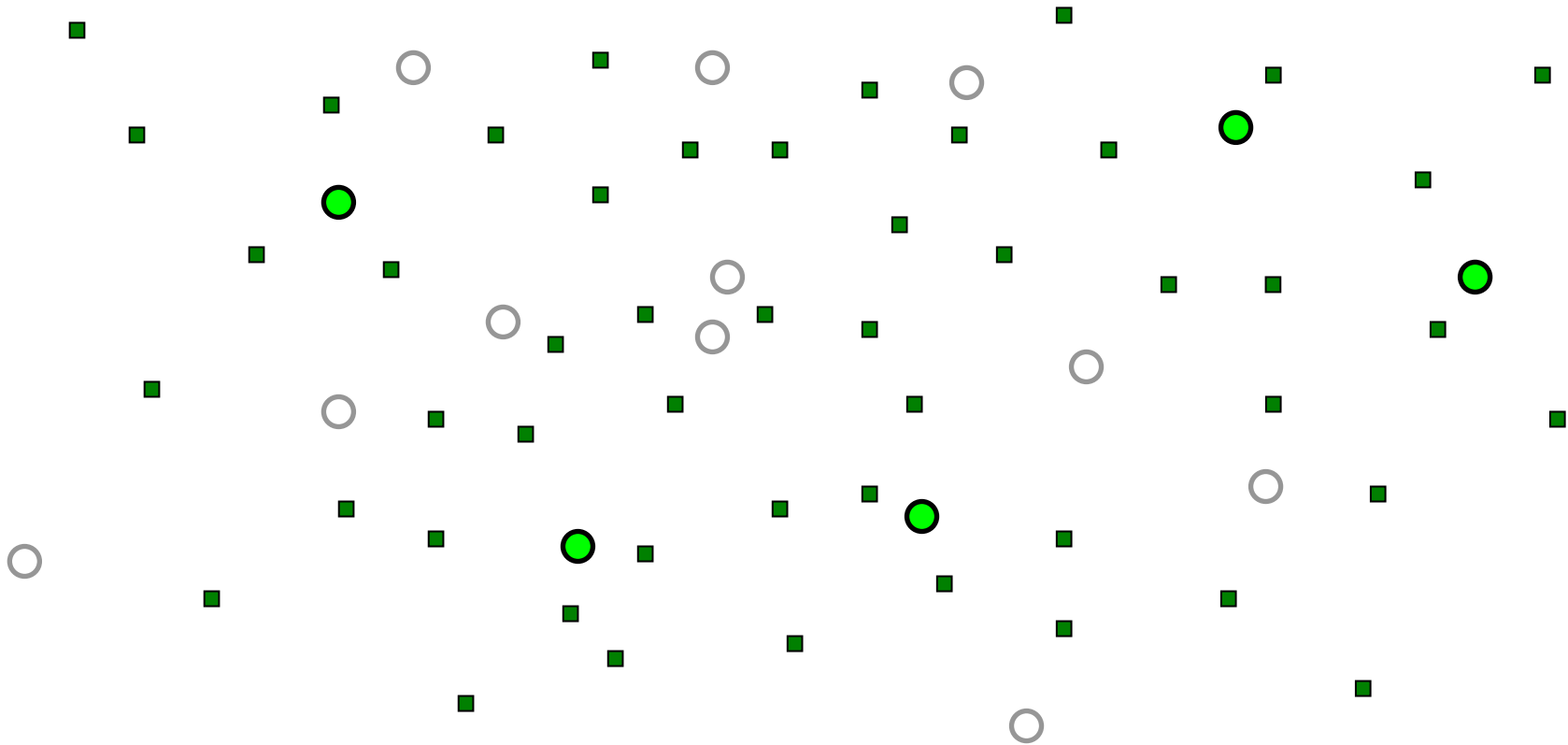
# Example (p-median)



assume  $p=5$

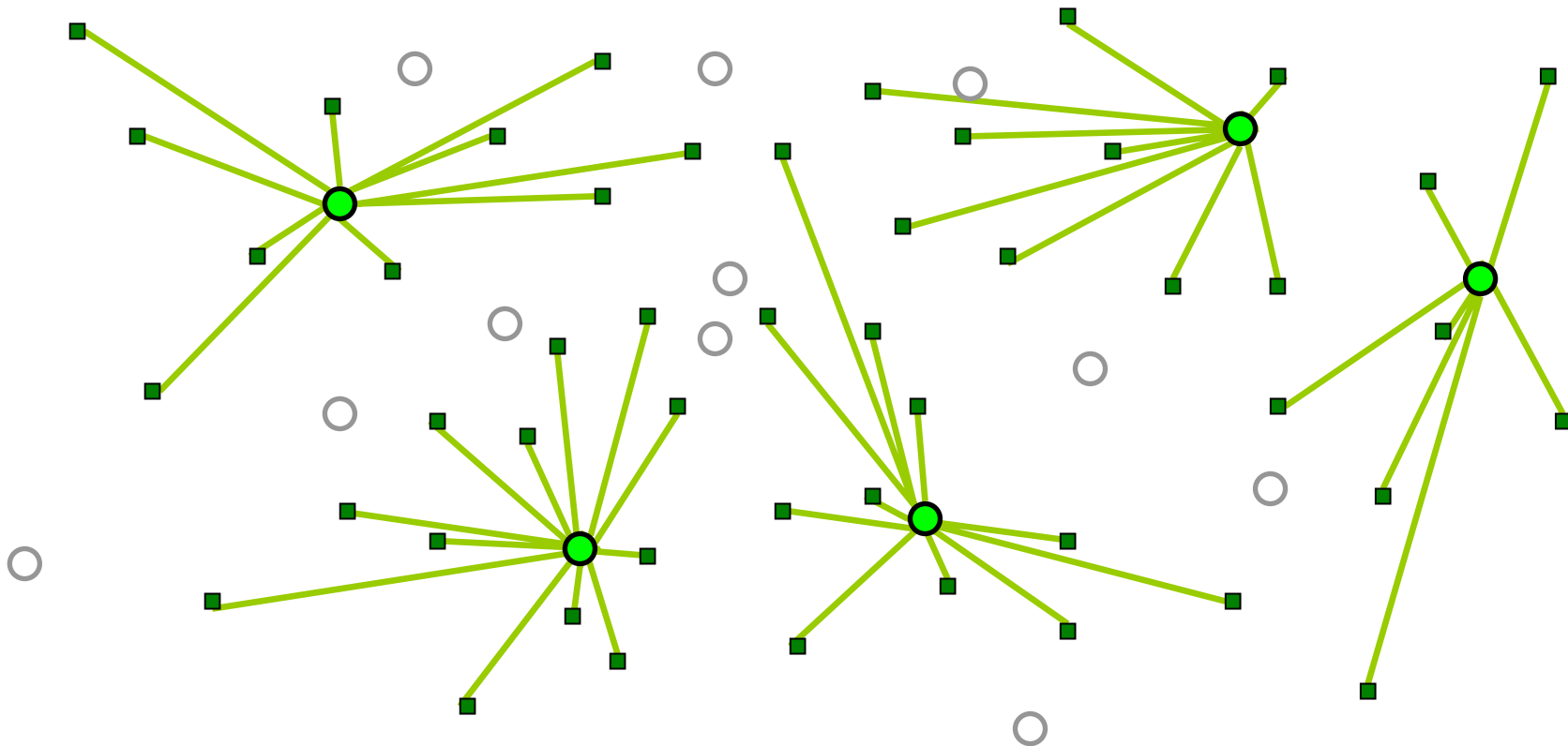
(5 facilities will be opened)

# Example (p-median)



This is a valid solution.

# Example (p-median)



This is a valid solution with the proper assignments.

# Our method

- The  $p$ -median problem is NP-hard.
- We use a hybrid GRASP metaheuristic:
  - “Greedy randomized adaptive search procedure”.
  - Multistart approach.
    - Each iteration:
      - Constructive algorithm;
      - Local search.
    - Intensification strategy:
      - *Path-relinking*. combines good solutions.
  - Finds near-optimal solutions for benchmark instances from the literature.
    - Bounds within 0.1% of best known for all instances tested.

# Footprint Optimization

- In our case:
  - each **exchange** is a p-median **user**:
    - 69,534 in total (all currently covered).
  - each **coordinate** is a p-median **facility**:
    - 1035 in total (all currently open).
  - Distances: network cost.
    - (PoP rate) · (hours used by exchange)
- With  $p=1035$ , we get the current network cost.
- We want the smallest  $p$  that preserves that cost.
  - Solve the p-median problem for various values of  $p$  to find best.
  - 700 was the value we found.



# Footprint Optimization

- With 700 PoPs (instead of 1035), potential savings on network cost:
  - Best-case scenario:
    - Everybody calls the cheapest (for AT&T) PoP available.
    - Monthly cost: \$3.357 million (unchanged)
  - Worst-case scenario:
    - Everybody calls the most expensive PoP available.
    - Monthly cost: reduced from \$3.604M to \$3.500 million.
    - Savings: up to \$104K a month, more than \$1.2M a year.
  - Average-case scenario:
    - Each customer equally likely to call all available PoPs.
    - Monthly cost: reduced from 3.424M to 3.414M.
    - Savings: up to 120K a year.

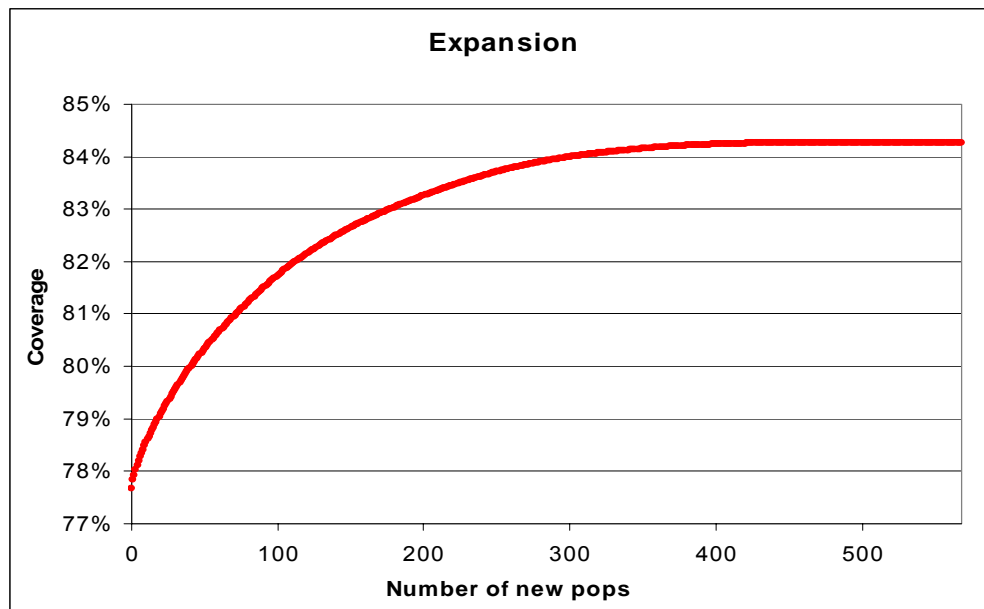
# Expanding the Footprint

- Second problem:
  - **Increase coverage beyond 77.66%.**
- AT&T can use UUNet PoPs:
  - 1,498 candidate PoPs.
  - 568 of those cover at least one new exchange.
- Main question:
  - If we want to open  $p$  new PoPs, which  $p$ ?
    - Goal: maximize coverage.
- This is the *maximum cover problem*:
  - It can be solved with the  $p$ -median tool.

# From Maximum Cover to p-median

- Idea: minimize number of customers *not covered*.
  - **Users:**
    - exchanges *not* currently covered.
  - **Facilities:**
    - all candidate UUNet PoPs;
    - dummy facility  $f_0$ .
  - **Distances:**
    - $d(u, f_i) = 0$ , if PoP  $i$  covers exchange  $u$ .
      - if  $u$  is covered, does not contribute to solution.
    - $d(u, f_0) = (\text{\#customers in exchange } u)$ ;
    - $d(u, f_i) = \text{infinity}$ , if PoP  $i$  does not cover  $u$ .
      - $u$  not covered: assigned to  $f_0$ , contributes to solution.
  - A dummy user can be used to ensure that  $f_0$  will always belong to the solution.

# Expansion



Coverage	Footprint
77.66%	current
78%	current+3
79%	current+19
80%	current+41
81%	current+72
82%	current+113
83%	current+177
84%	current+301
84.27%	current+464

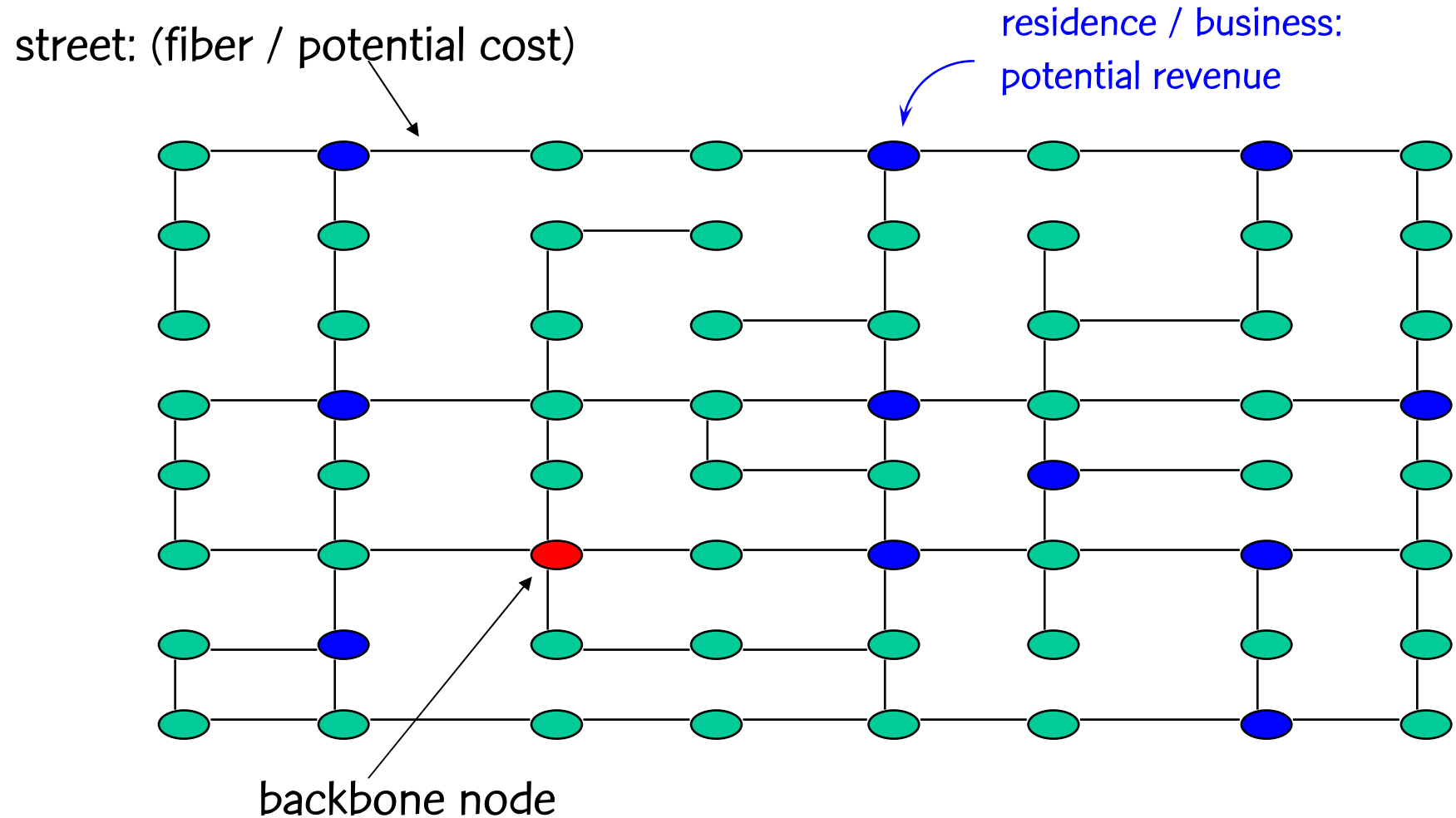
# Application 2:

## Local access network design

# Local access network design

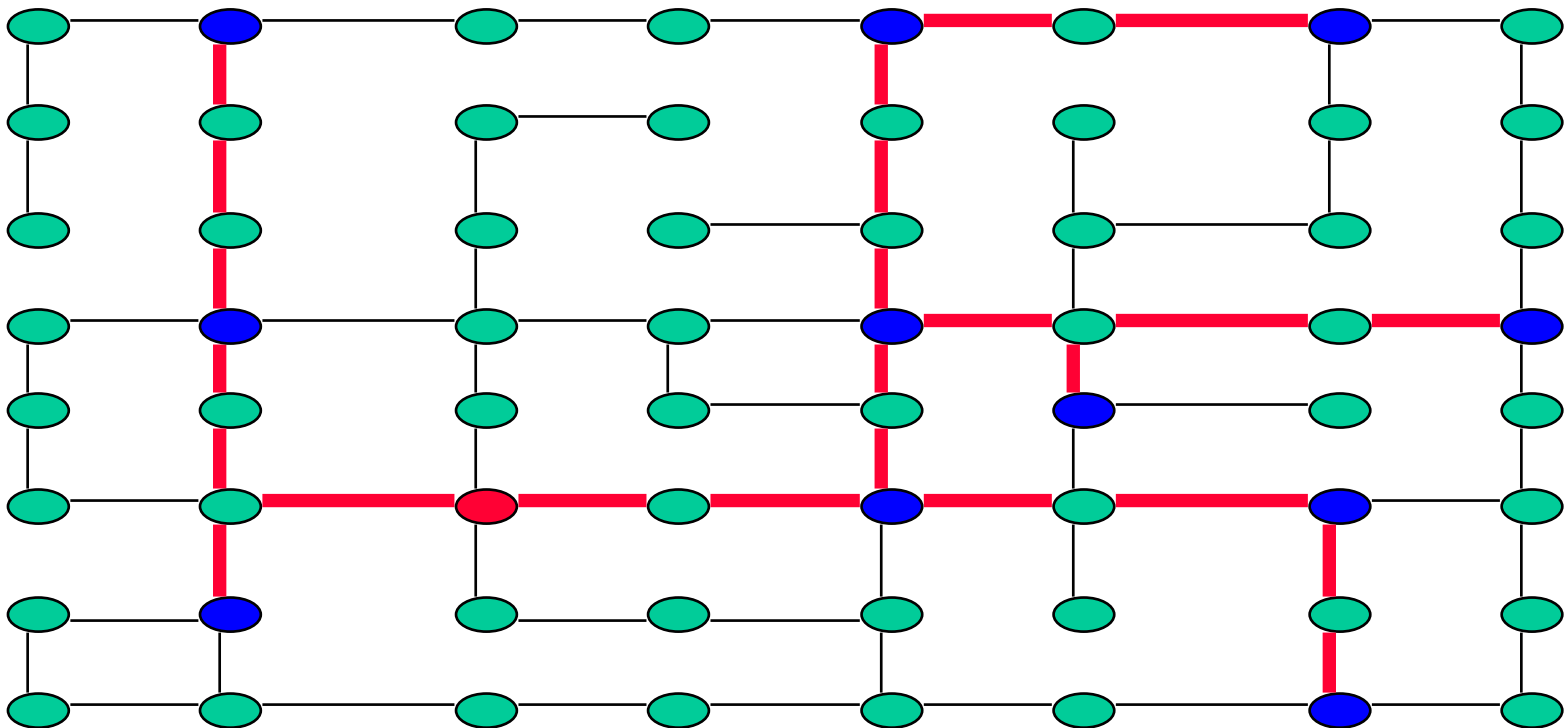
- Design a local access network taking into account tradeoff between:
  - cost of network
  - revenue potential of network

# Local access network design



# Solve prize collecting Steiner tree problem

max prize collected minus edge cost

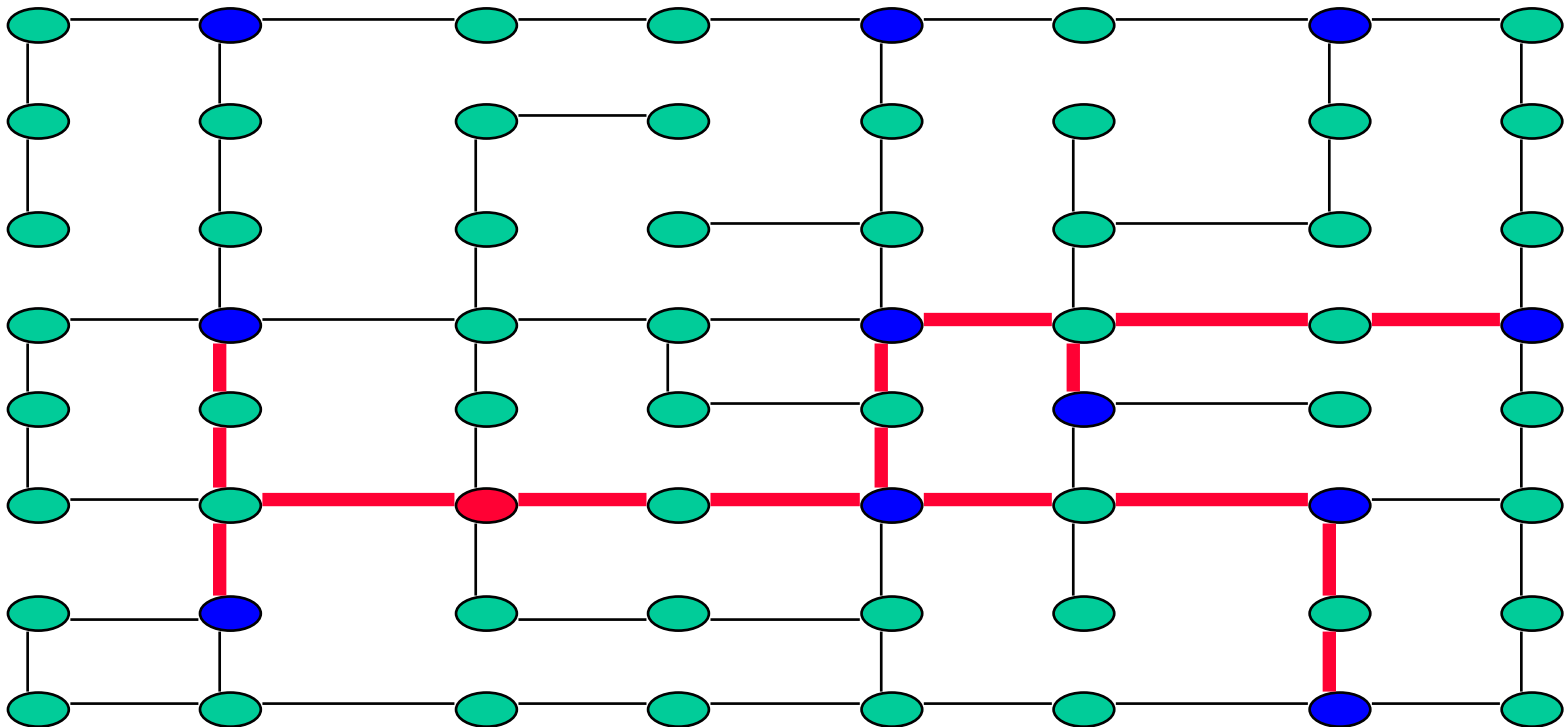


Here all prizes are collected.



# Solve prize collecting Steiner tree problem

max prize collected minus edge cost



Here not all prizes are collected.

# Solve prize collecting Steiner tree problem

- Typical dimension: 20,000 to 100,000 nodes.
- Compute lower bounds with cutting planes algorithm of Lucena & Resende (Discrete Applied Math., 2003)
- Compute solutions (upper bounds) with GRASP with path-relinking of Canuto, Resende, & Ribeiro (Networks, 2001)

# Application 3:

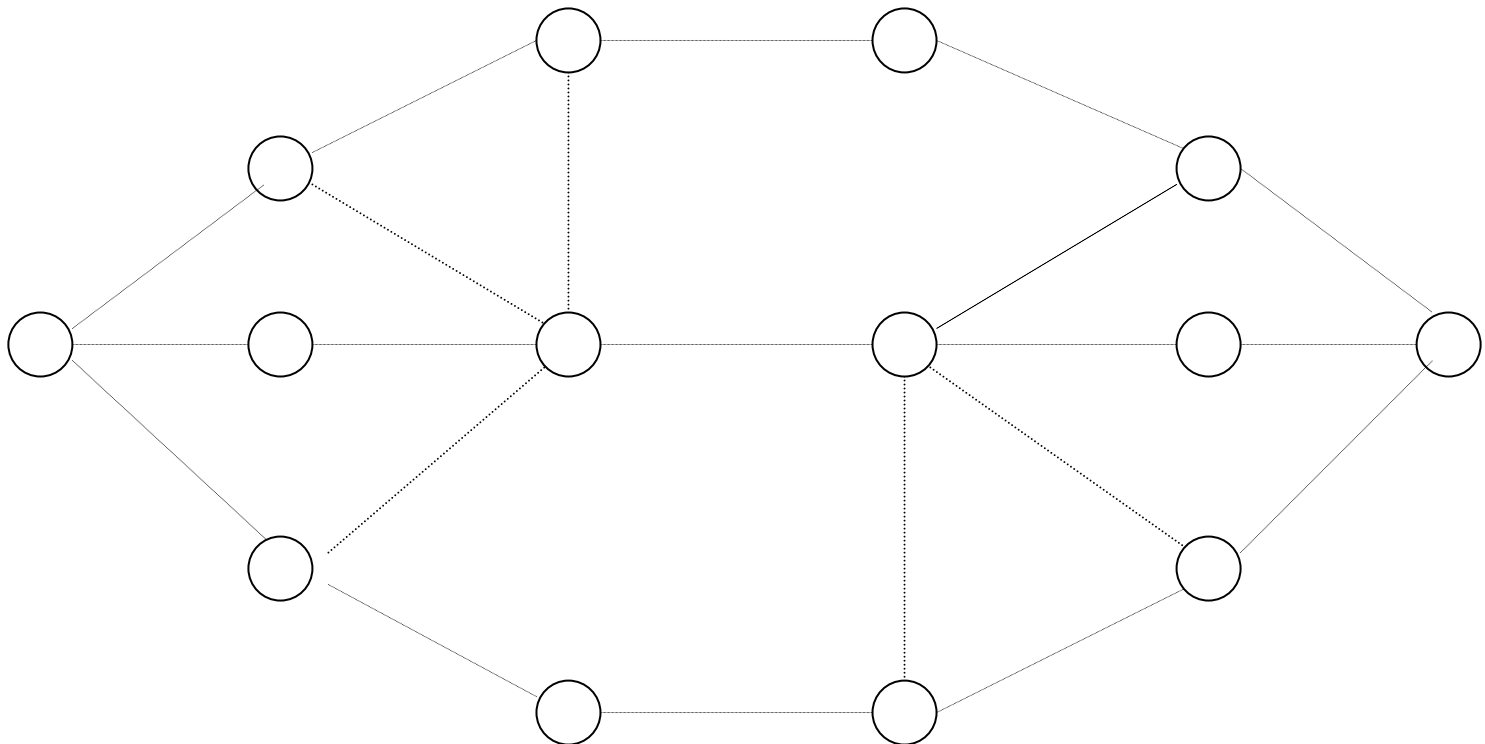
## Routing Frame Relay Permanent Virtual Circuits (PVC)

# Routing Frame Relay Permanent Virtual Circuits (PVC)

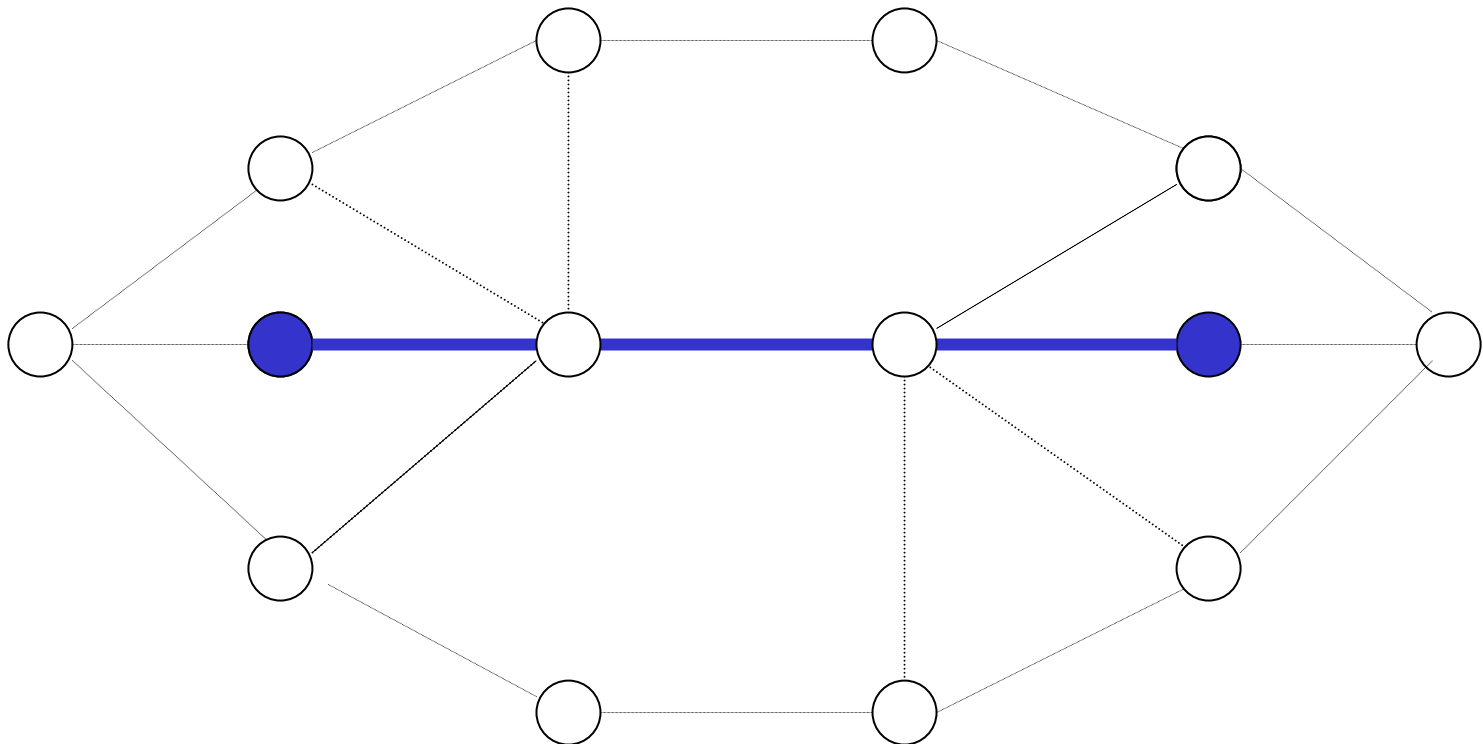
Resende & Ribeiro (Networks, 2003)

- Frame relay (FR) service
  - provides virtual private networks to customers
  - by provisioning a set of permanent (long-term) virtual circuits (PVC) between customer endpoints on the backbone network
- Provisioning of PVCs
  - routing is done either automatically by switch or by network designer without any knowledge of future requests
  - over time these decisions cause inefficiencies in network and occasional rerouting of PVCs is needed

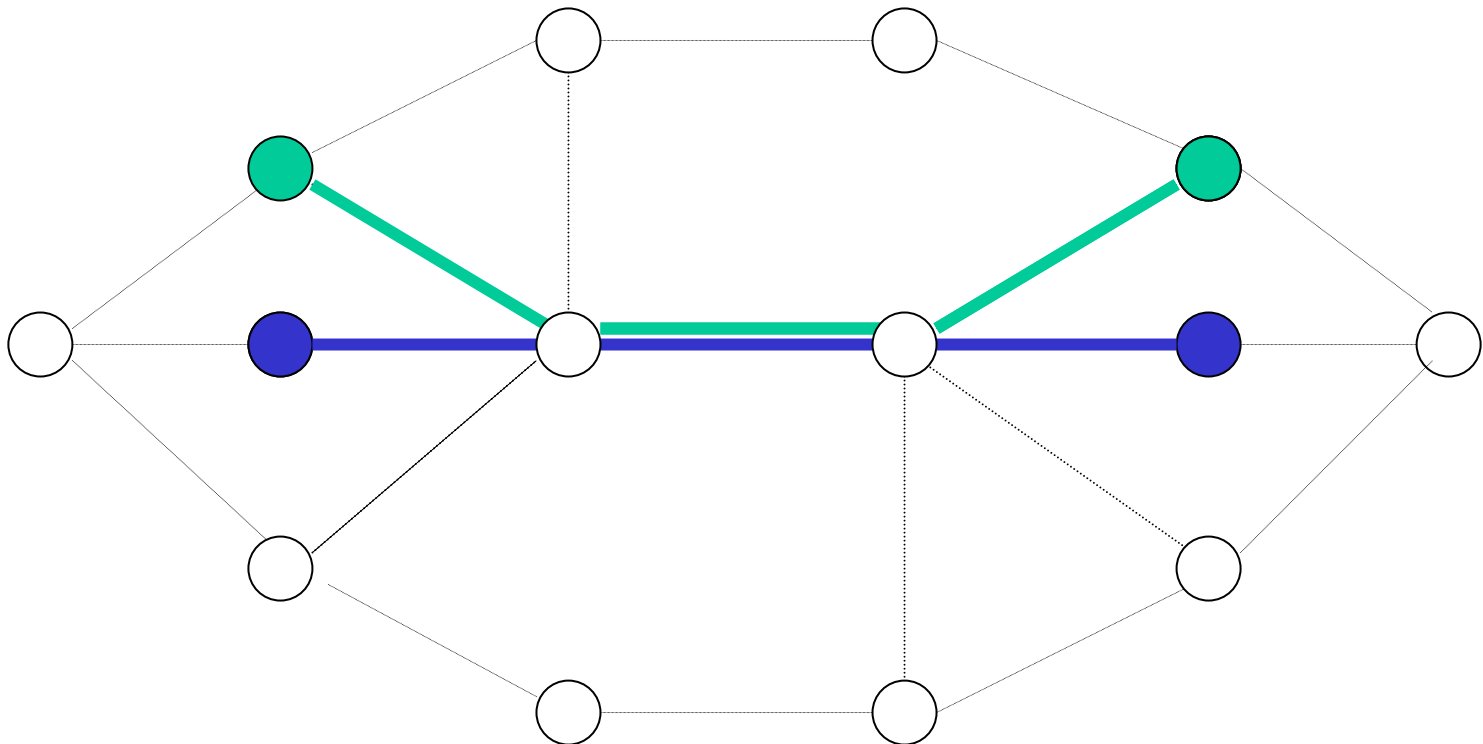
# PVC routing: example



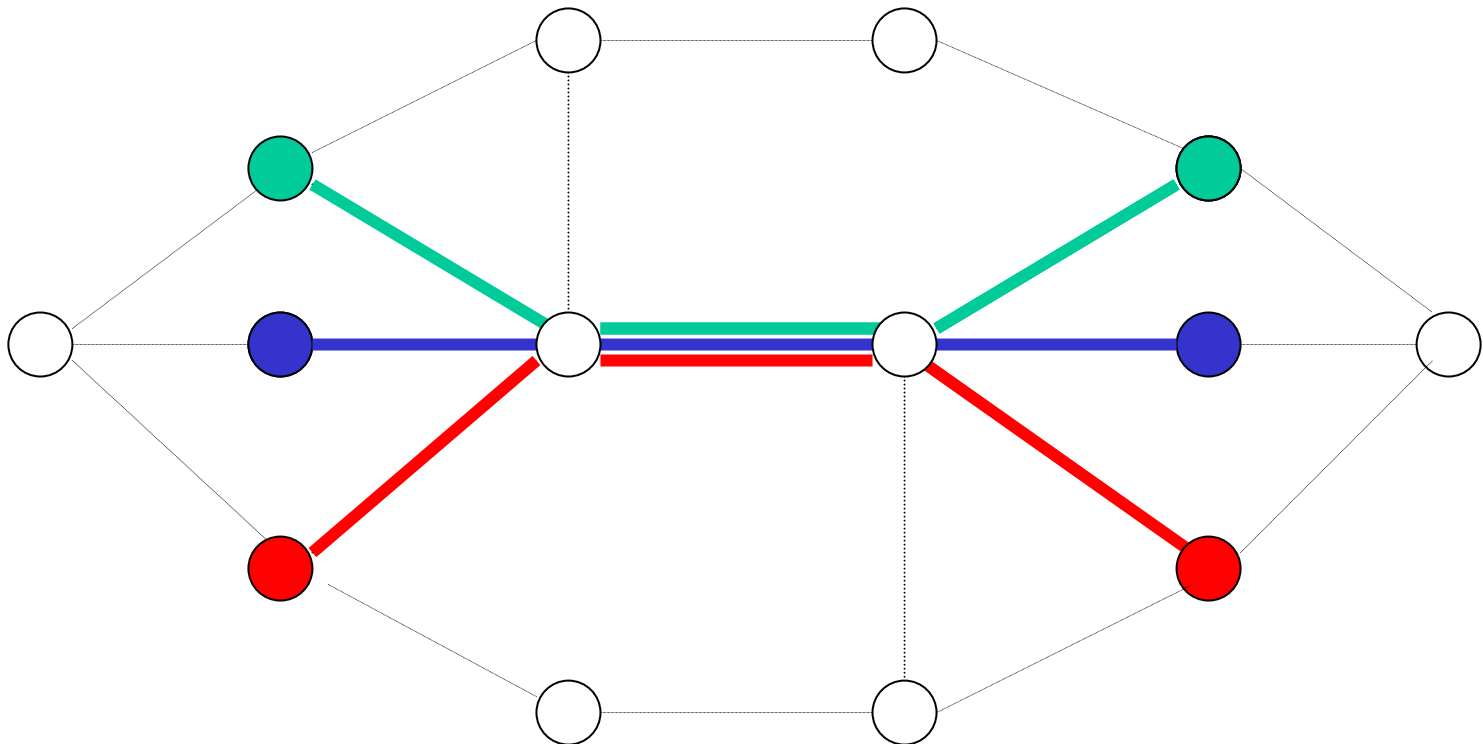
# PVC routing: example



# PVC routing: example

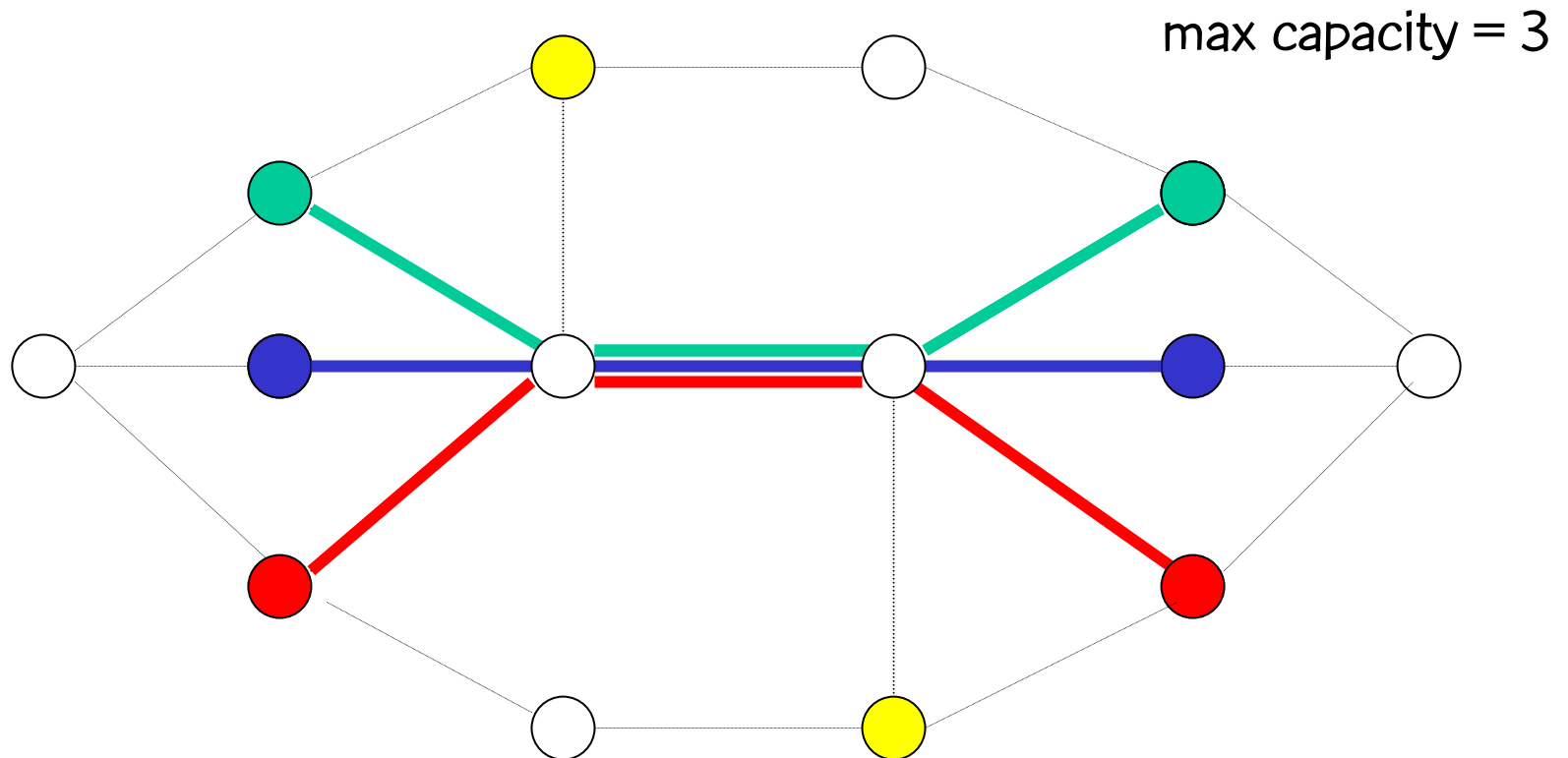


# PVC routing: example

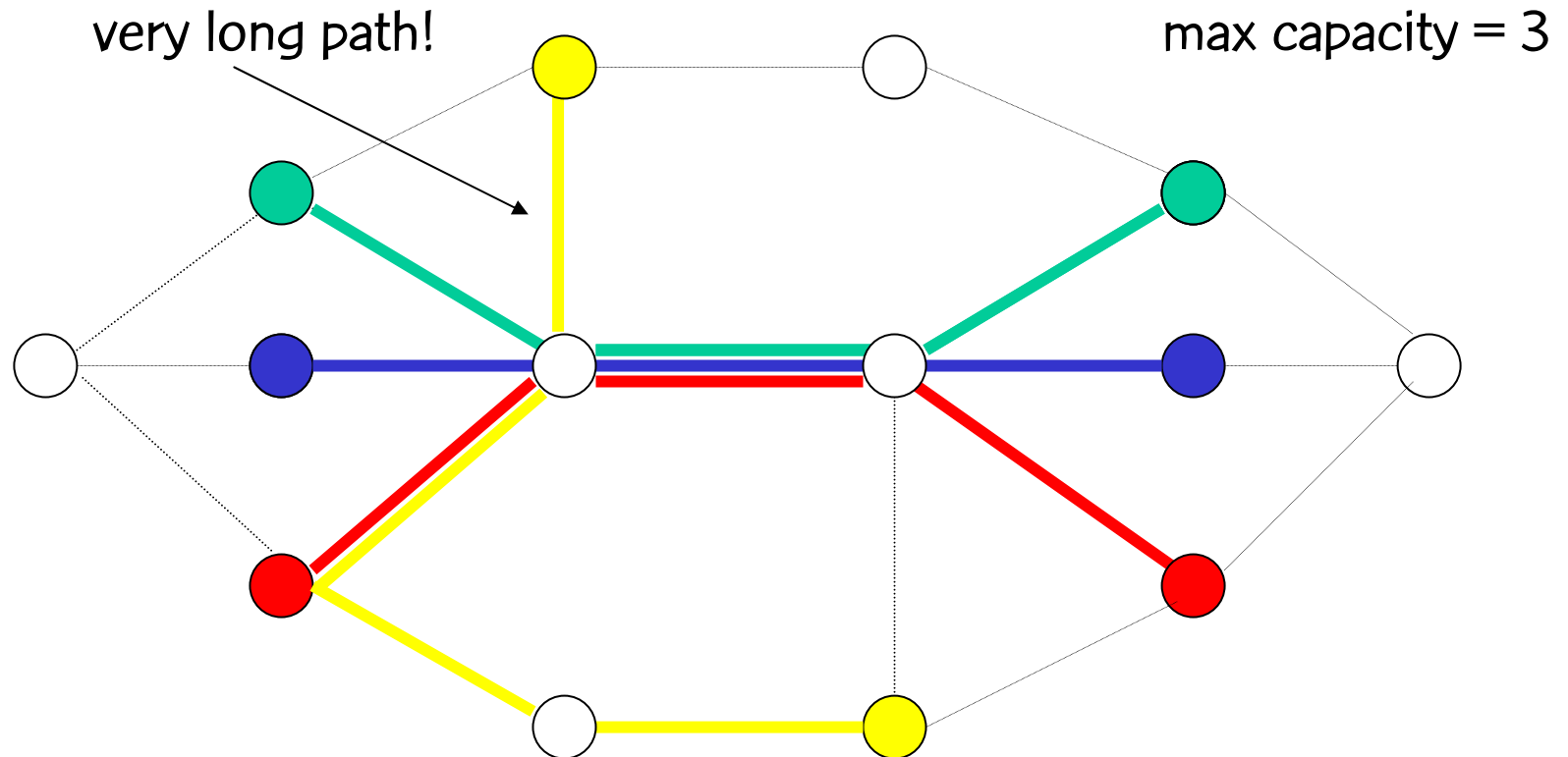




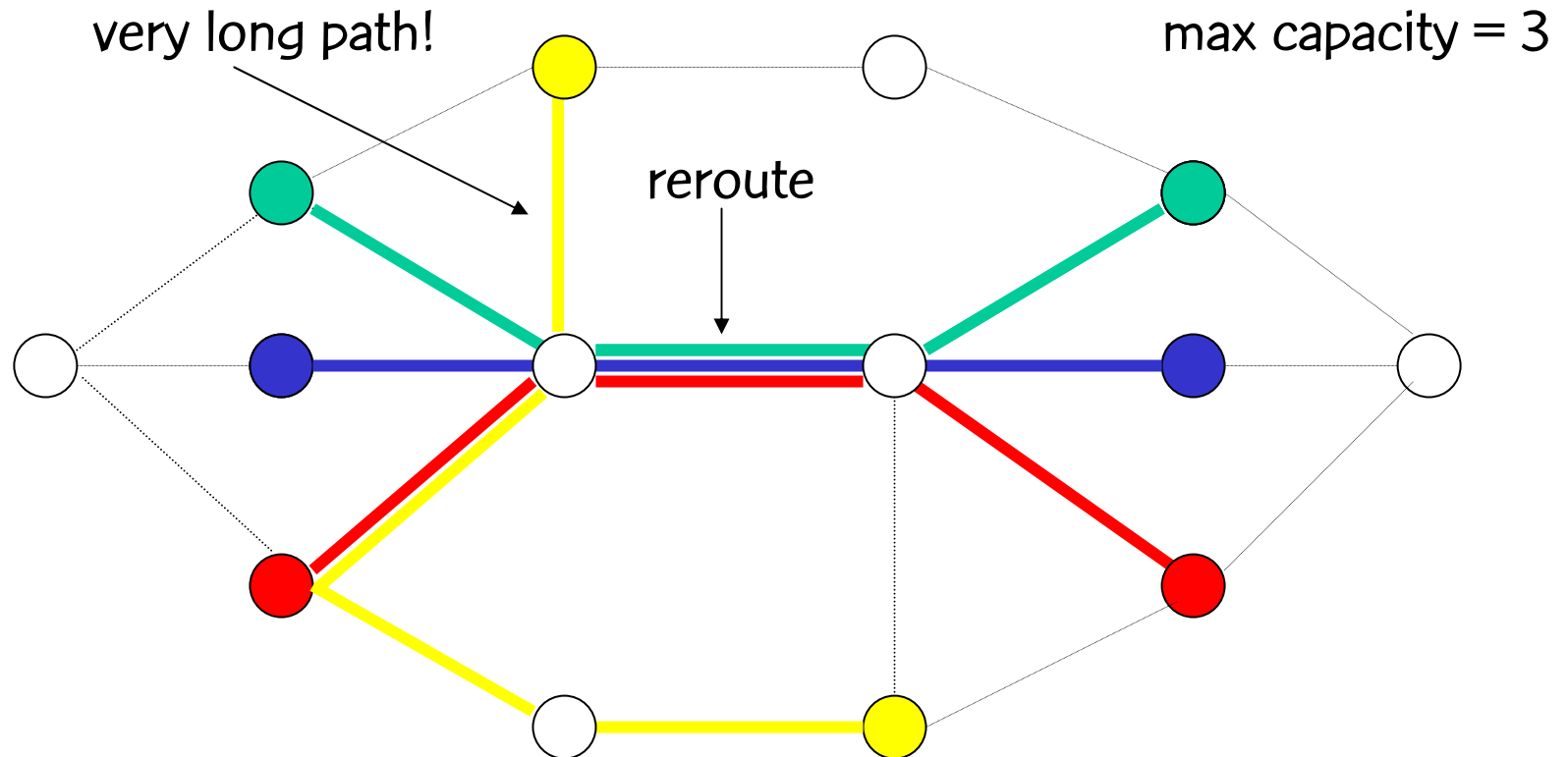
# PVC routing: example



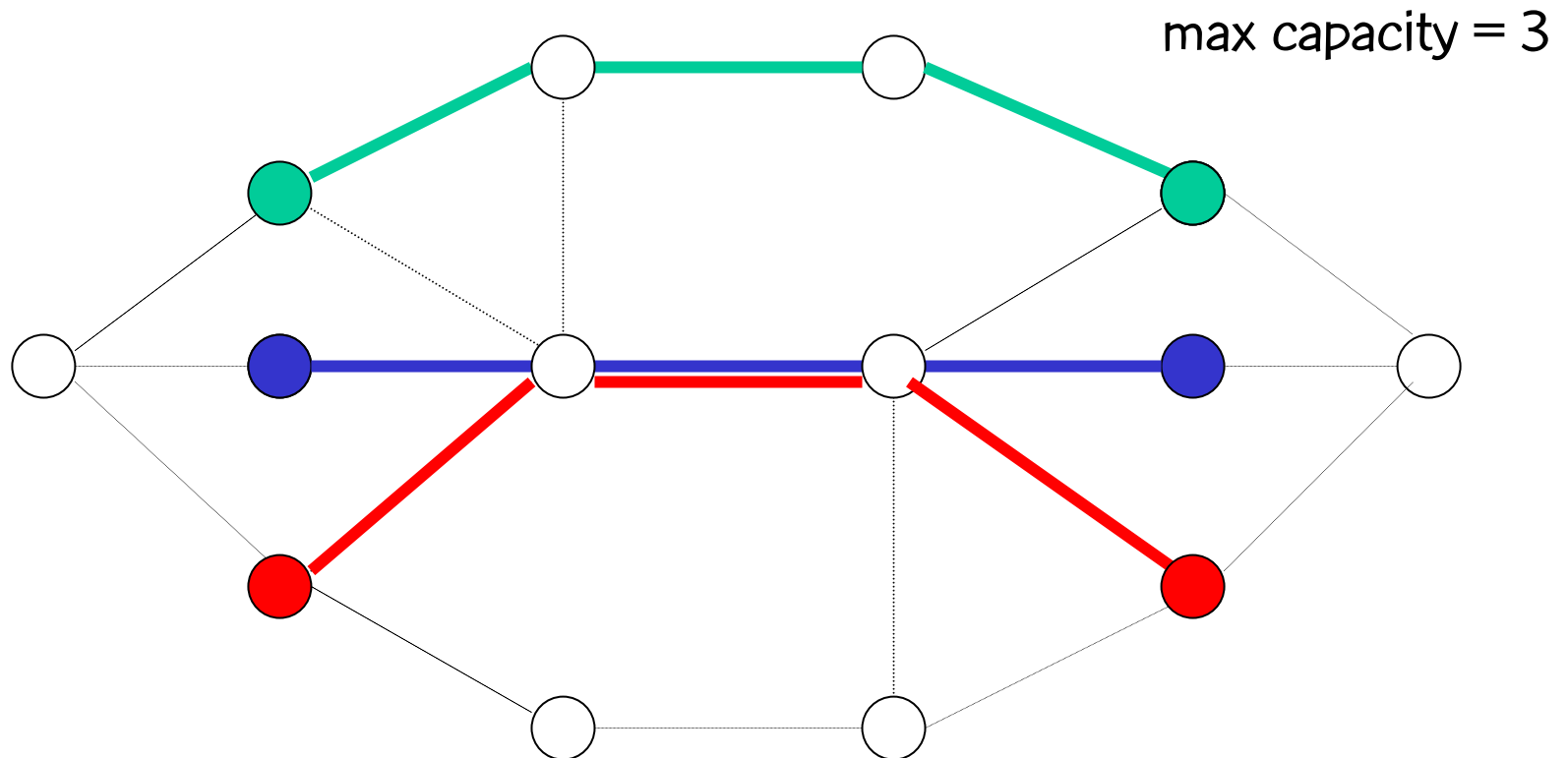
# PVC routing: example



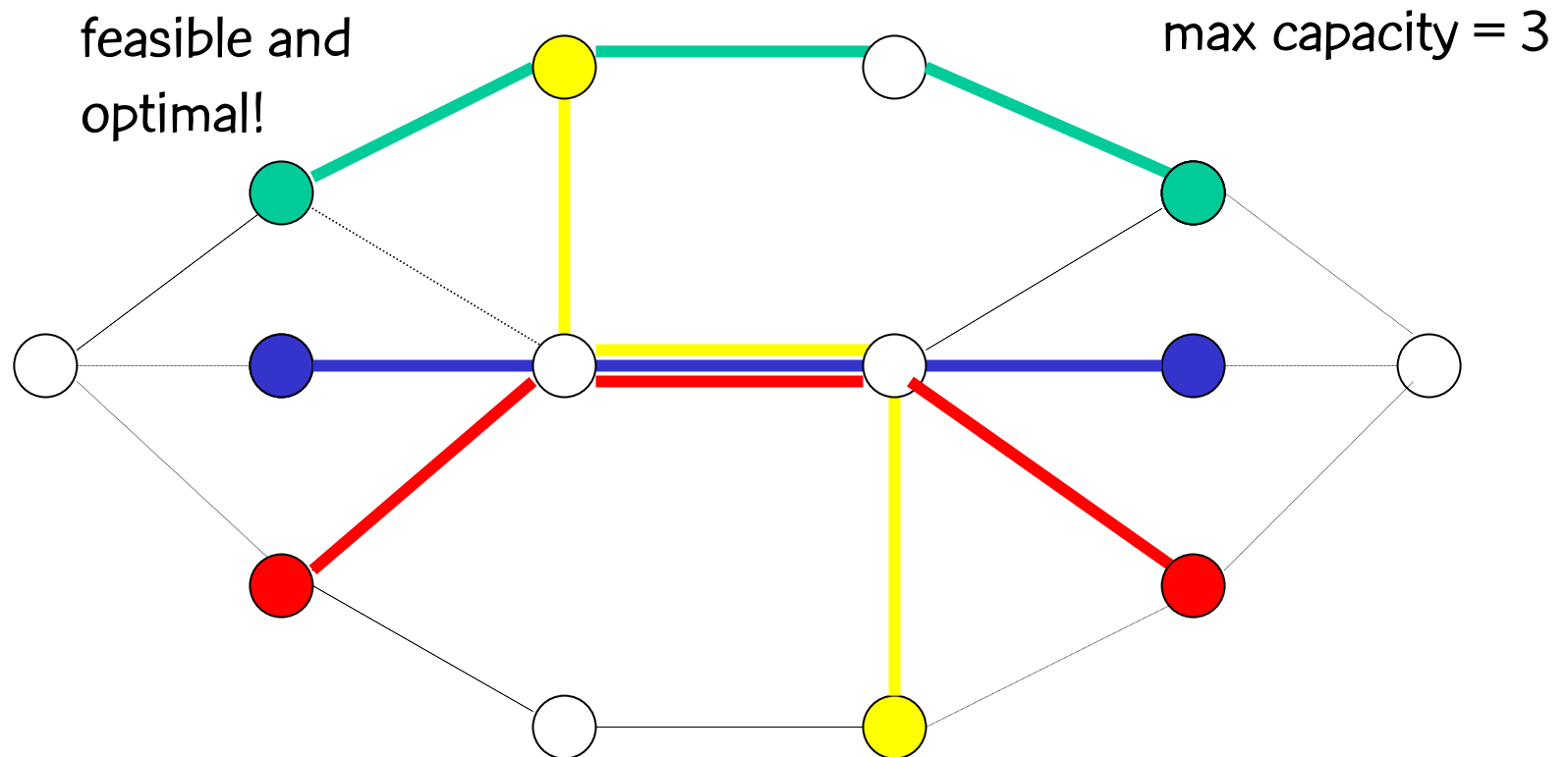
# PVC routing: example



# PVC routing: example



# PVC routing: example



# Routing Frame Relay Permanent Virtual Circuits (PVC)

- one approach is to order PVCs and apply algorithm on FR switch to reroute
  - however, taking advantage of factors not considered by FR switch routing algorithm may lead to greater efficiency of network resource utilization
  - FR switch algorithm is typically fast since it is also used to reroute in case of switch or trunk failures
  - this can be traded off for improved network resource utilization when routing off-line

# FR PVC Routing Problem

- given undirected FR network  $G = (V, E)$ , where
  - $V$  denotes  $n$  backbone nodes (FR switches)
  - $E$  denotes  $m$  trunks connecting backbone nodes
- for each trunk  $e = (i, j)$  let
  - $b(e)$  be the bandwidth (max kbits/sec rate) of trunk  $e$
  - $c(e)$  be the max number of PVCs that can be routed on trunk  $e$
  - $d(e)$  be the propagation and hopping delay associated with trunk  $e$

# FR PVC Routing Problem

- list of demands (or commodities  $K = \{1, \dots, p\}$ ) is defined by
  - origin - destination pairs
  - $r(p)$  - effective bandwidth requirement (forward, backward, overbooking) for PVC  $p$
- objective is to minimize
  - delays
  - network load unbalance
- subject to
  - technological constraints



# FR PVC Routing (bandwidth packing) Problem

- route for PVC ( $o$ ,  $d$ ) is
  - sequence of adjacent trunks
  - first trunk originates in node  $o$
  - last trunk terminates in node  $d$
- set of routing assignments is feasible if for all trunks  $e$ 
  - total PVC bandwidth requirements routed on  $e$  does exceed  $b(e)$
  - number of PVCs routed on  $e$  is not greater than  $c(e)$

# Mathematical programming formulation

$$\min \phi(x) = \sum_{(i,j) \in E, i < j} \phi_{i,j}(x_{i,j}^1, \dots, x_{i,j}^p, x_{j,i}^1, \dots, x_{j,i}^k)$$

subject to

$$\sum_{k \in K} r_k (x_{i,j}^k + x_{j,i}^k) \leq b_{i,j}, \quad \forall (i,j) \in E, i < j$$

$$\sum_{k \in K} (x_{i,j}^k + x_{j,i}^k) \leq c_{i,j}, \quad \forall (i,j) \in E, i < j$$

$$\sum_{(i,j) \in E} x_{i,j}^k - \sum_{(i,j) \in E} x_{j,i}^k = \begin{cases} 1, & \text{if } i \in V \text{ is source for } k \in K \\ -1, & \text{if } i \in V \text{ is destination for } k \in K \\ 0, & \text{otherwise} \end{cases}$$

$$x_{i,j}^k \in \{0,1\}, \quad \forall (i,j) \in E, \forall k \in K.$$

$x_{i,j}^k = 1$ , iff trunk  $(i,j)$   
is used to route  
PVC  $k$ .

# Cost function

- Linear combination of
  - delay component
  - load balancing component

- Delay component:  $d_{i,j} \sum_{k \in K} \rho_k (x_{i,j}^k + x_{j,i}^k)$

# Cost function: Load balancing component

- We use the measure of Fortz & Thorup (2000) to compute congestion:

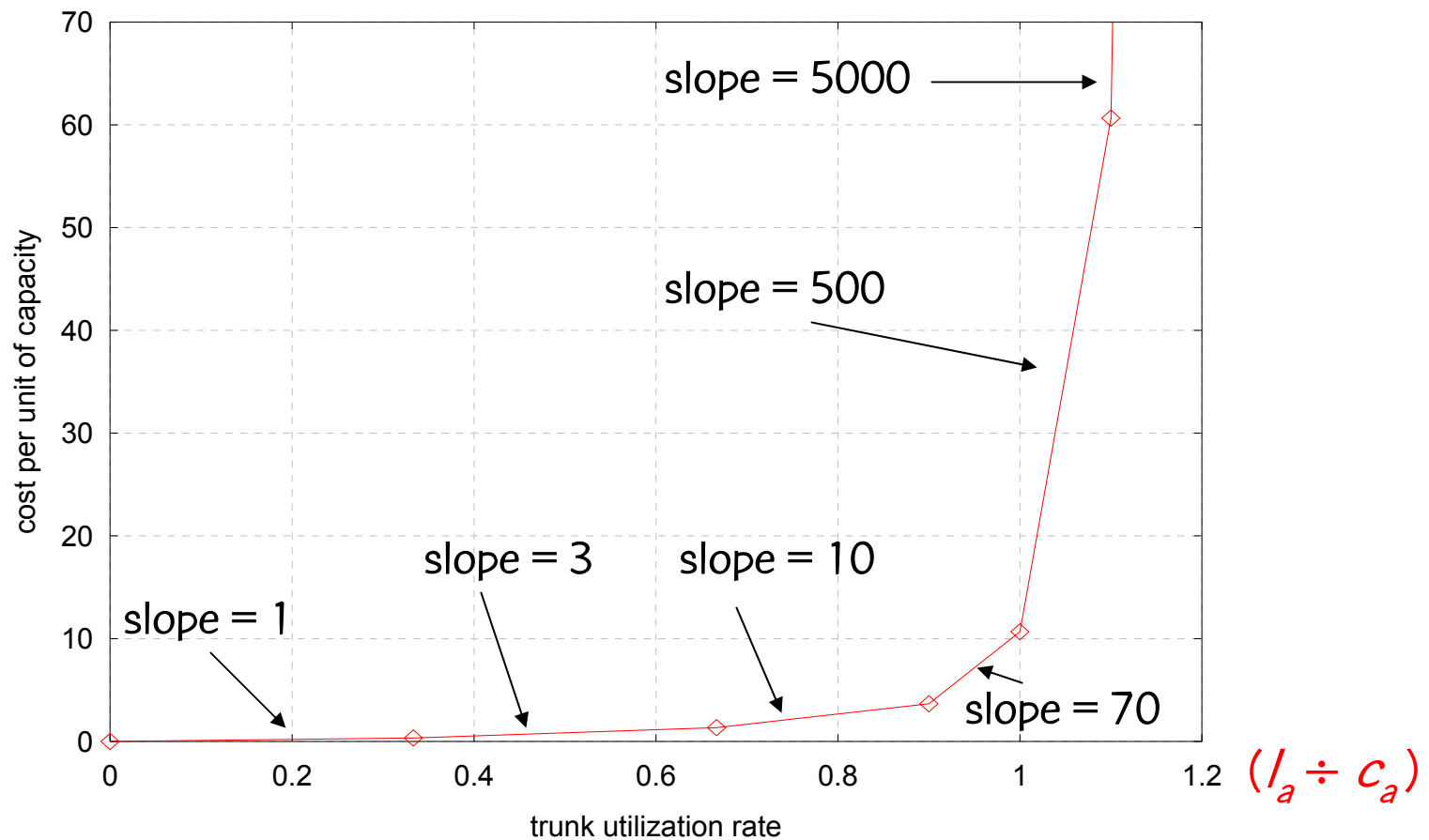
$$\Phi = \Phi_1(l_1) + \Phi_2(l_2) + \dots + \Phi_{|E|}(l_{|E|})$$

where  $l_e$  is the load on link  $e \in E$ ,

$\Phi_e(l_e)$  is piecewise linear and convex,

$\Phi_e(0) = 0$ , for all  $e \in E$ .

# Piecewise linear and convex $\Phi_e(I_e)$ link congestion measure



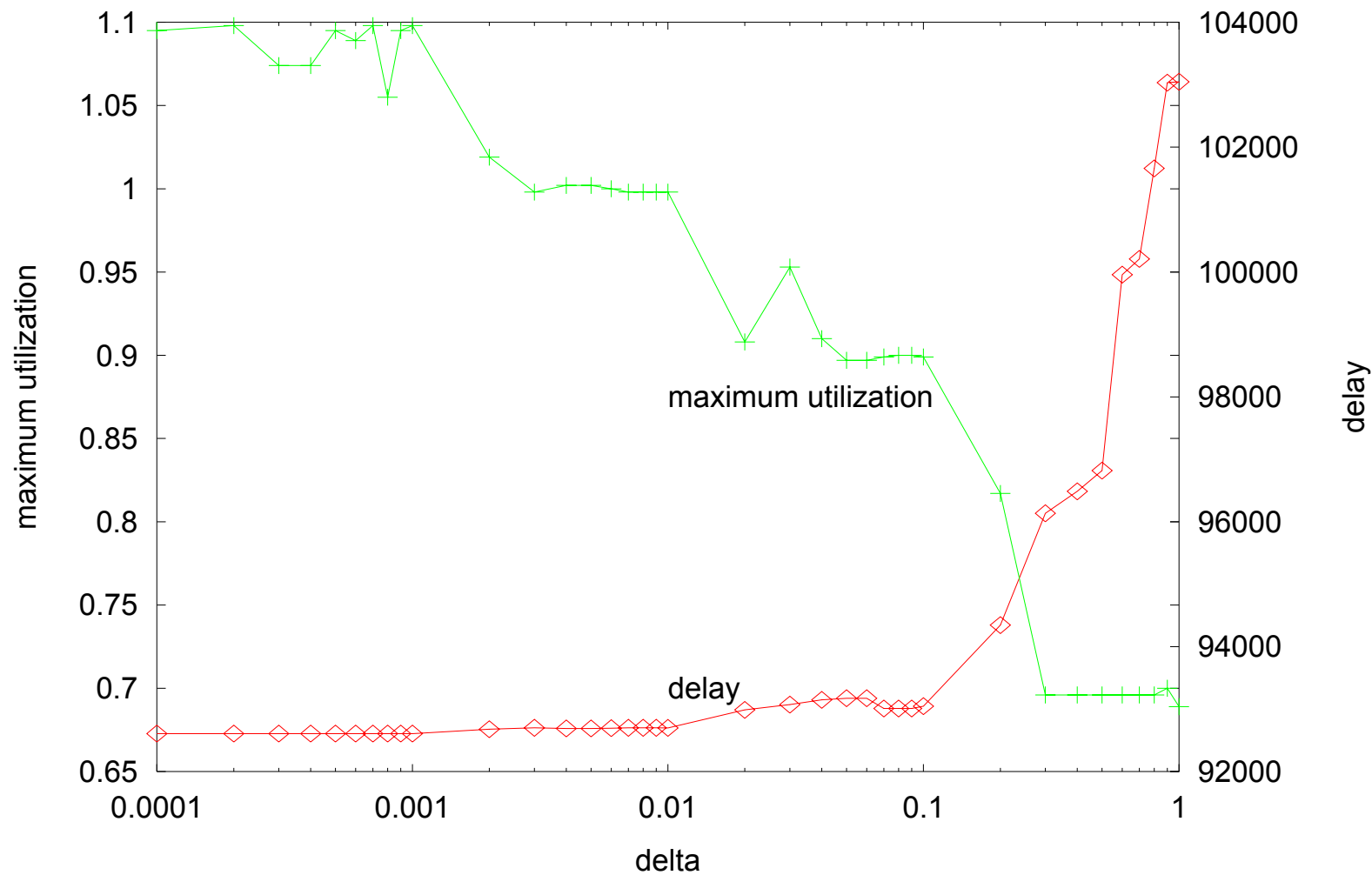
# Solution method

- GRASP

- **Construct** by choosing unrouted pair, biasing in favor of high bandwidth requirement. Use shortest path routing using as edge distance the incremental cost associated with routing  $r_k$  additional units of demand on edge  $(i, j)$ .
- **Local search**: for each PVC  $k \in K$ , remove  $r_k$  units of flow from each edge in its current route, compute incremental edge weights, and reroute.

- Path-relinking

- moves are route changes (target solution route replaces current solution route)



# Application 4:

## Mining for cliques in telephone call detail database



# Mining for cliques in telephone call detail database

Abello, Pardalos, & Resende (1999); Abello, Resende, & Sudarsky (2002)

- Data explosion
- Massive graphs arising from telephone call detail database
- Structure of call detail graph
- Searching for large cliques and bicliques
- Some experimental results

# Data explosion

(Abello, Pardalos, & Resende, Eds., "Handbook of Massive Data Sets," Kluwer, 2002)

- Proliferation of massive data sets brings with it computational challenges
- Data avalanche arises in a wide range of scientific and commercial applications
- Today's data sets are of high dimension and are made up of huge numbers of observations:
  - More often they overwhelm rather than enlighten
- Outstripped the capabilities of traditional data measurement, data analysis, and data visualization tools

# Data explosion

- A variety of massive data sets can be modeled as a very large multi-digraph
  - Special set of edge attributes represent special characteristics of application
- WWW: nodes are pages, edges are links pointing from one page to another
- Telephone call graph is another example ...

# Call detail

- Every phone call placed on AT&T network generates a record (~ 200 bytes) with:
  - Originating & terminating numbers
  - Start time & duration of call
  - Other billing information
- The collection of these records is known as the **Call Detail Database**

# Call detail

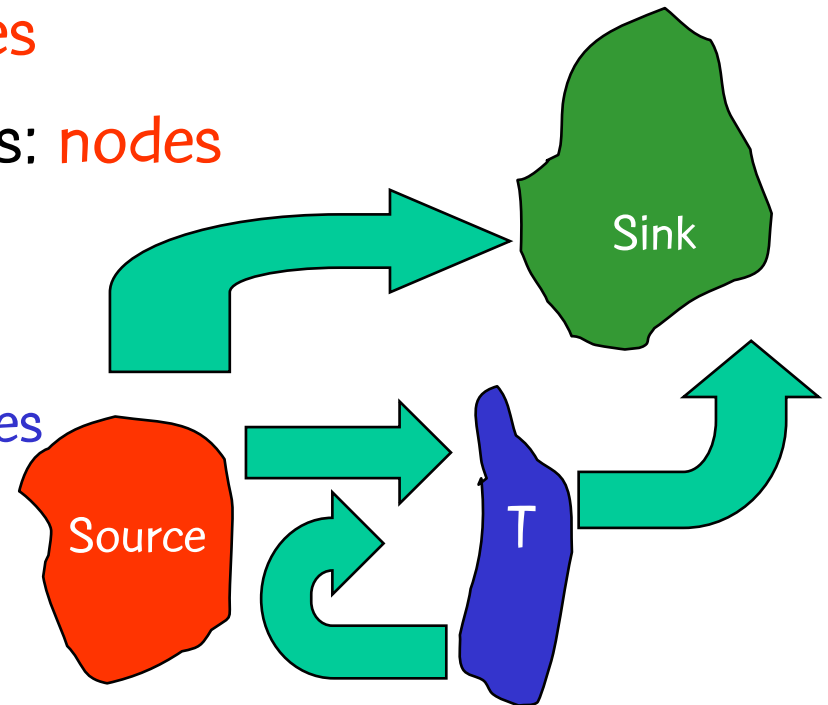
- AT&T system (in 2000) generated:
  - 250 million records per day (on average)
  - 320 million records on busy day
  - 18 terabytes of data per year
- Data is accessed for:
  - Billing & customer inquiries
  - Marketing & traffic engineering

# Call detail graph

- $G = (V, E)$  is a directed graph:
  - $V$  is the set of phone numbers
  - $E$  is the set of phone calls
    - $(u, v) \in E$  implies that phone  $u$  called phone  $v$
- $G$  quickly grows into a huge graph
  - Hundreds of millions of nodes and billions of edges
  - Our goal is to work with one year of data ( $\sim 1$  Tb)

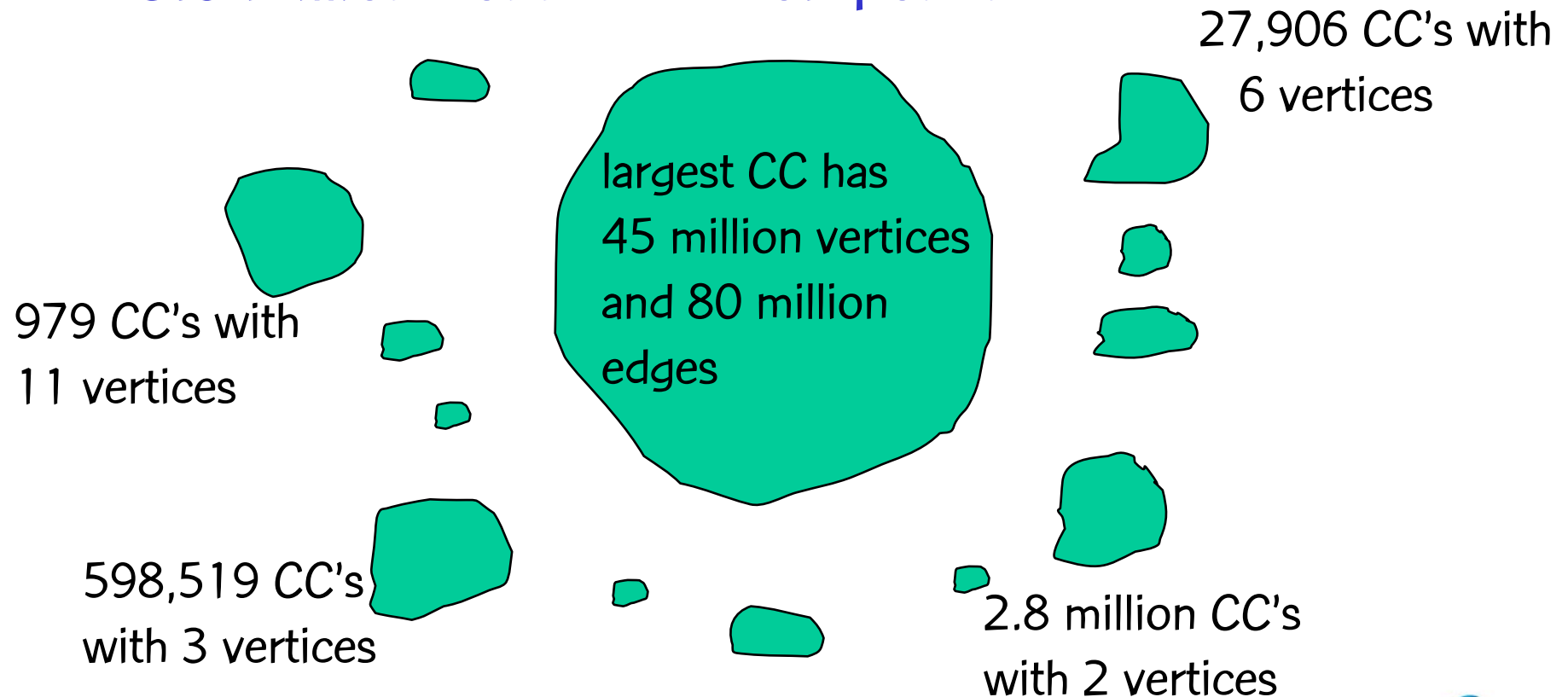
# Structure of call detail graph

- Consider a 12-hour call detail graph
  - 123 million records: **edges**
  - 53 million phone numbers: **nodes**
    - 21 million **source nodes**
    - 22 million **sink nodes**
    - 10 million **transmittal nodes**



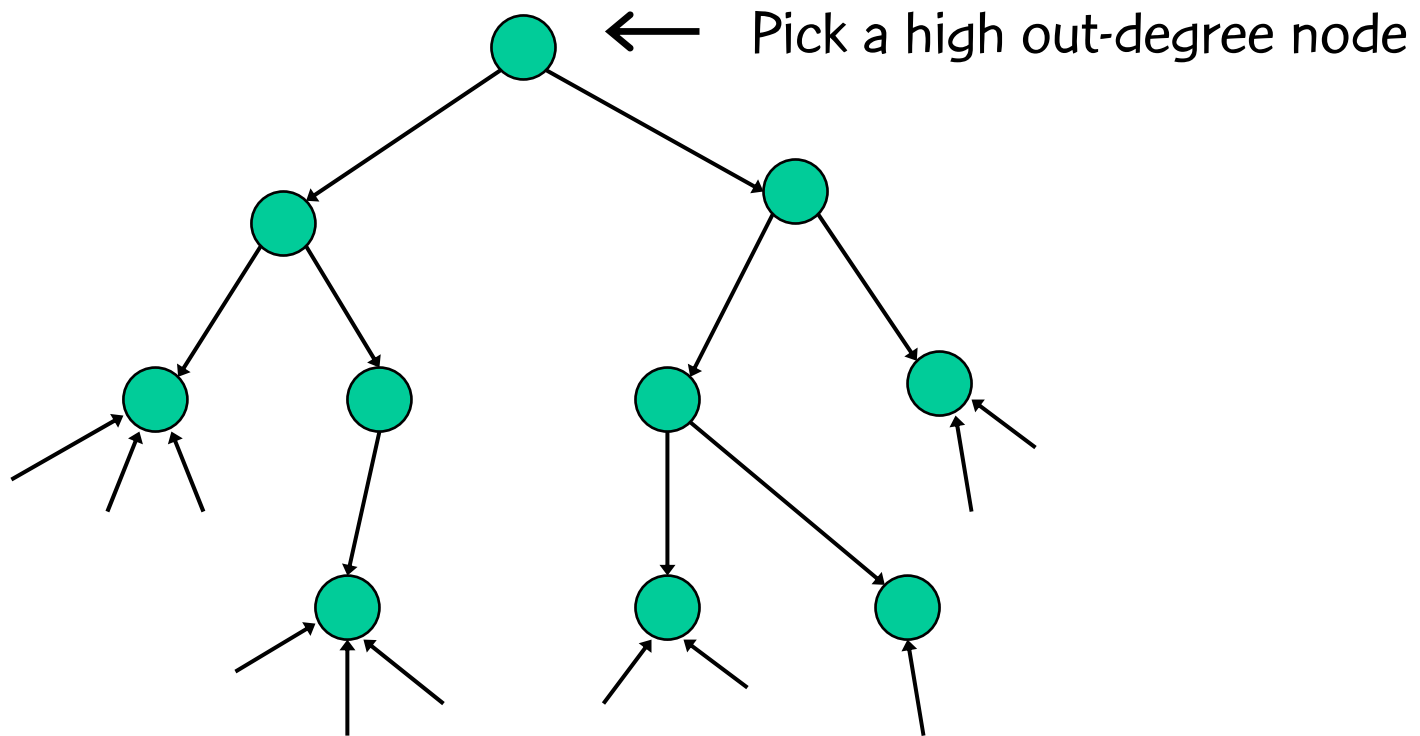
# Connected components

3.6 million connected components

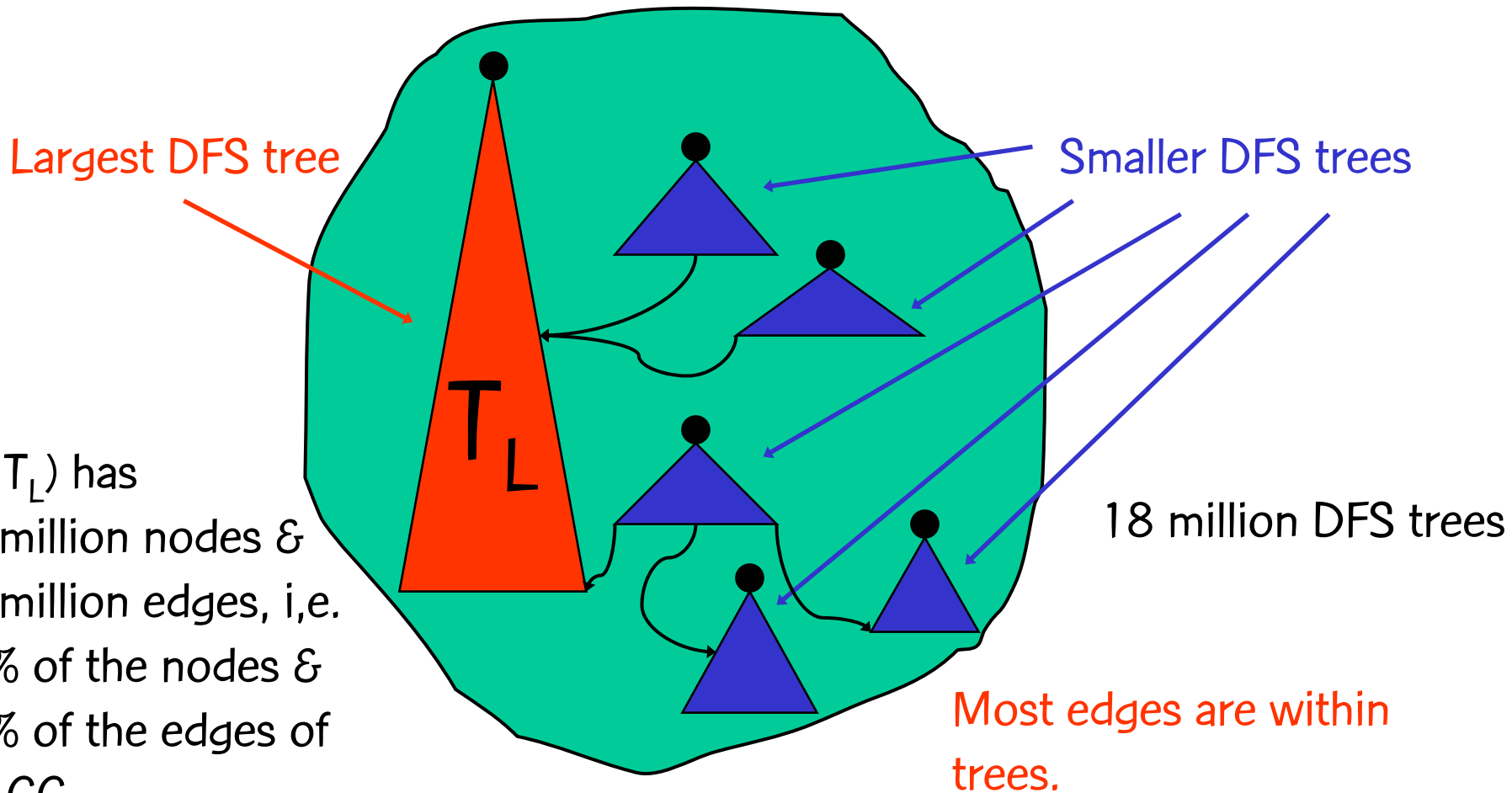




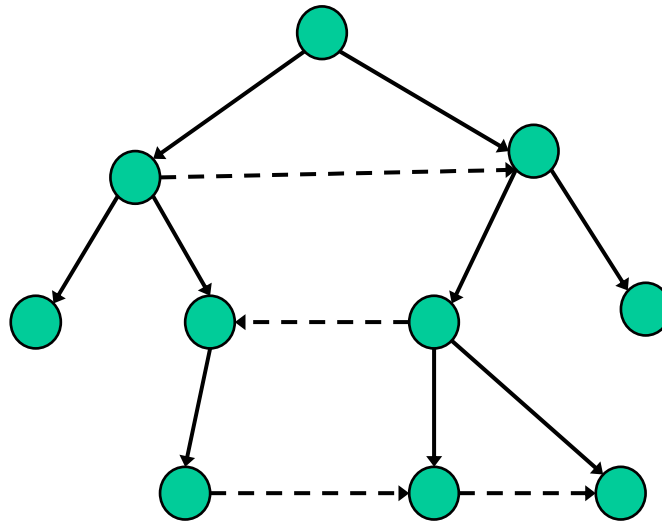
# Depth first search (DFS) tree



# DFS trees in largest CC



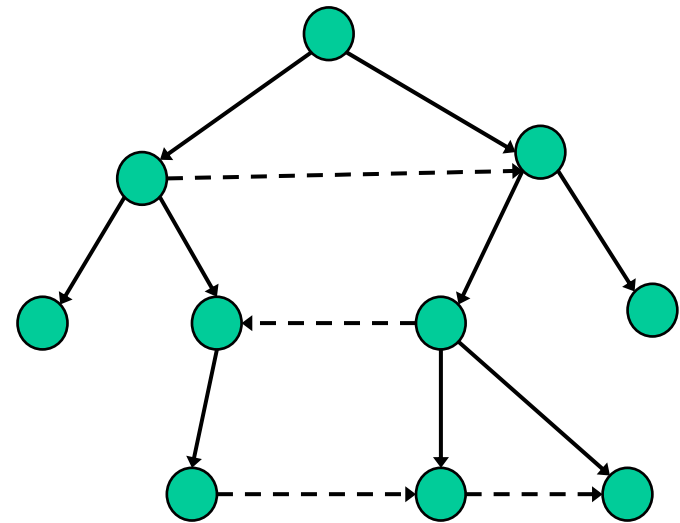
# Subgraph induced by DFS tree nodes



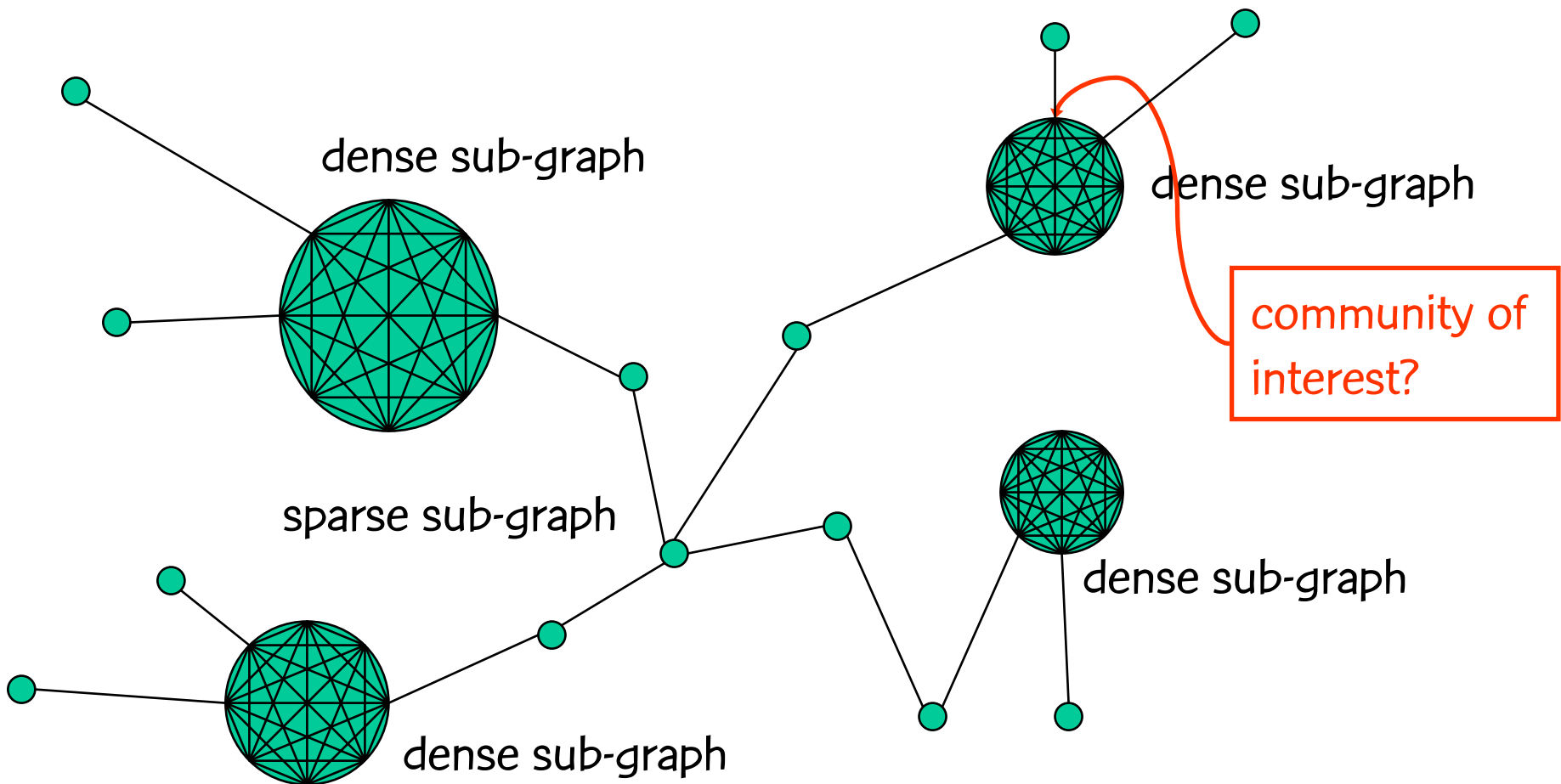
- Most subgraphs induced by DFS tree nodes are very sparse:  $|E| < \log(|V|)$
- Few are dense:  $|E| > \sqrt{|V|}$  with at most 32 nodes

# Dense subgraphs

- Dense subgraphs could be
  - within  $G$  (DFS tree)
  - among different  $G$  (DFS tree)
- Counting edges:
  - most are within  $G$  (DFS tree)
  - leaves few edges between different  $G$  (DFS tree)

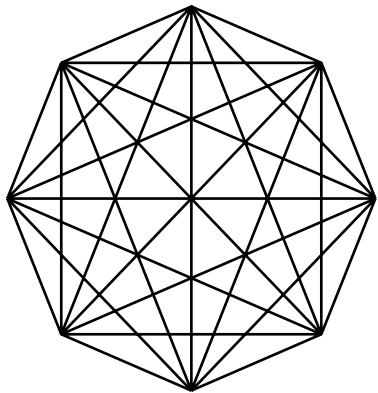


# Macro structure of call detail graph

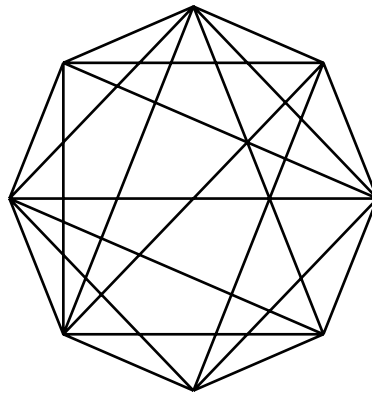


# Searching for dense subgraphs

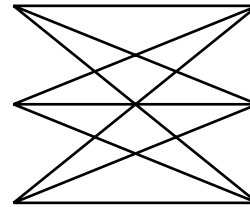
- We look for two types of subgraphs
  - cliques or quasi-cliques
  - bicliques or quasi-bicliques



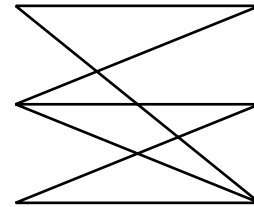
clique



quasi-clique



biclique



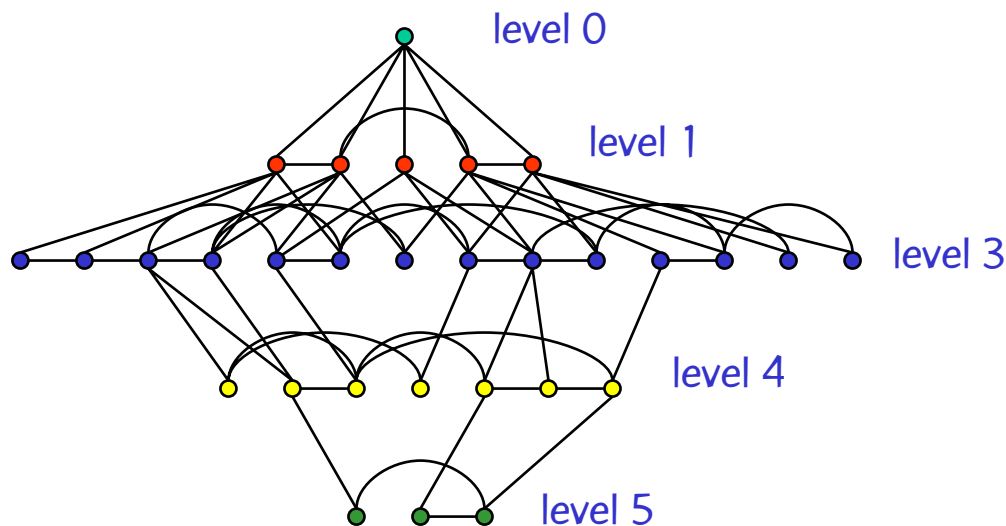
quasi-biclique

# Clique case

- We illustrate the approach with the clique case.
  - We work on connected component of transmittal nodes (no cliques in sources or sinks)
  - Breadth first search decomposition
  - Peeling off vertices to focus in on large cliques
  - Finding cliques in a subgraph

# Breadth first search decomposition

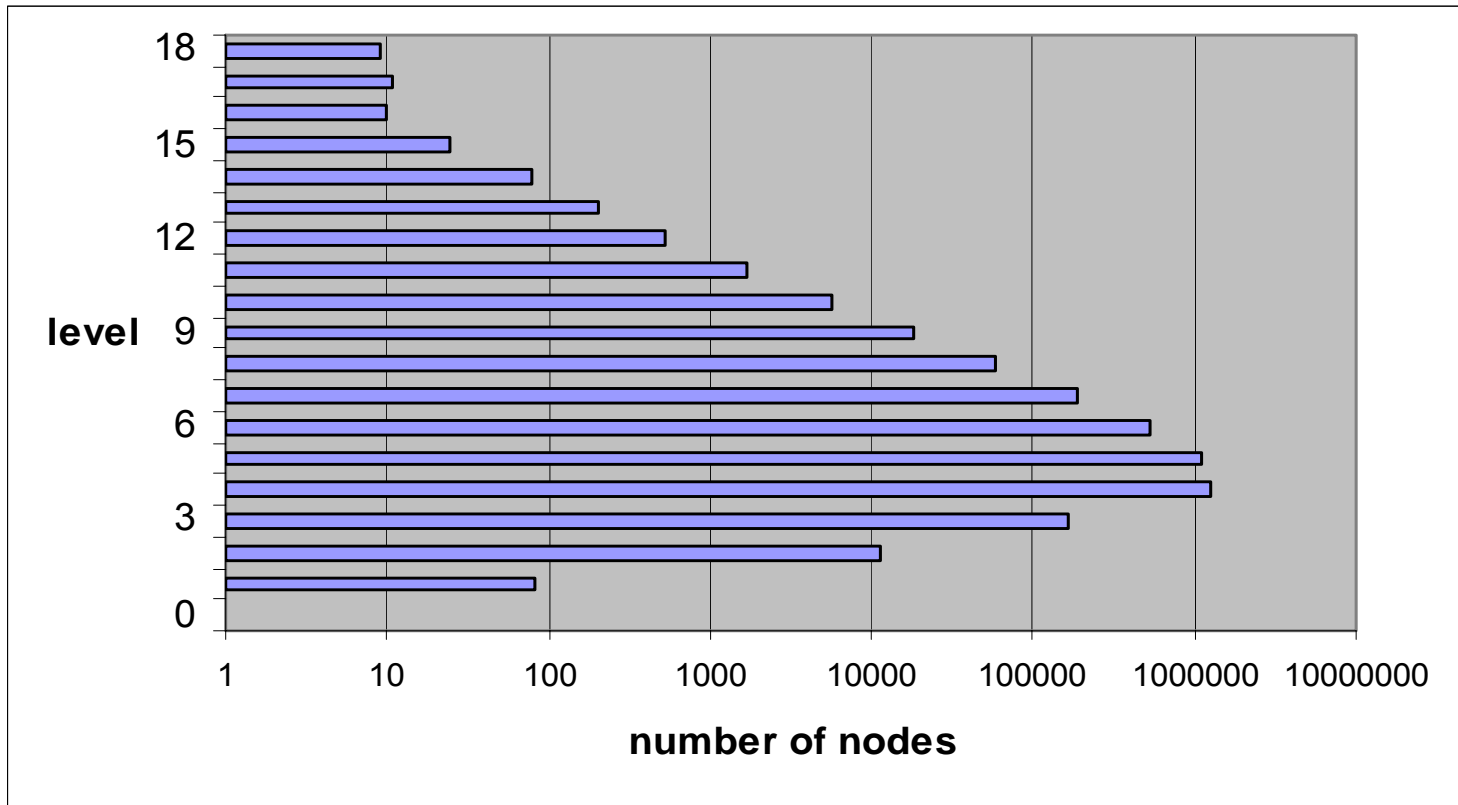
- Given a graph  $G$  one can decompose its vertices into levels



There are no cliques spanning three or more levels.

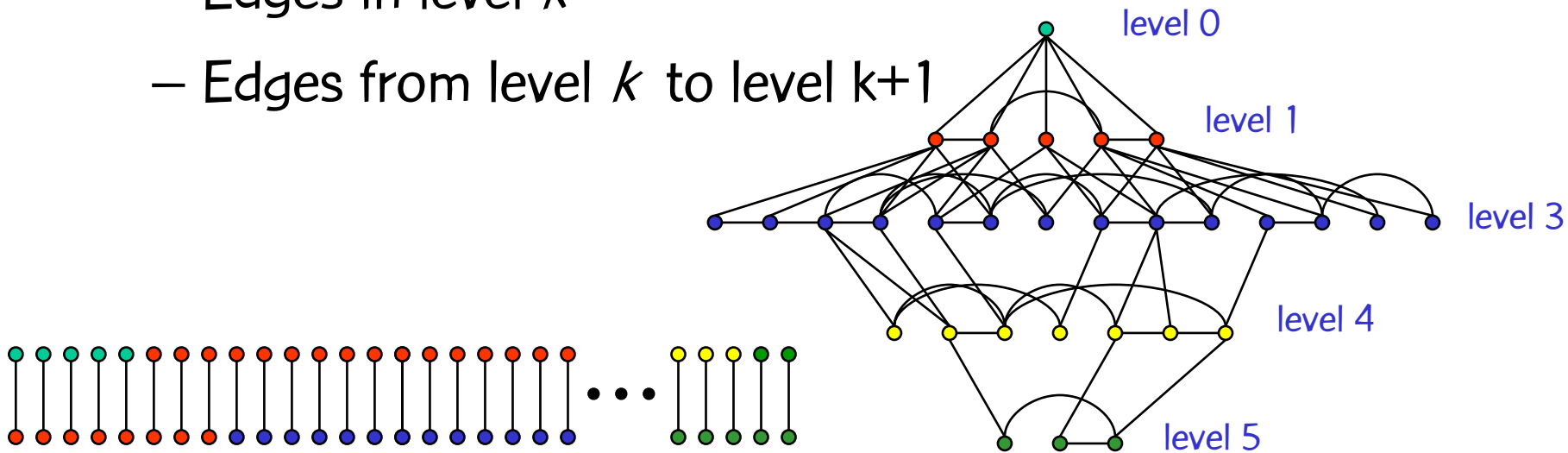


# BFS: distribution of nodes per level



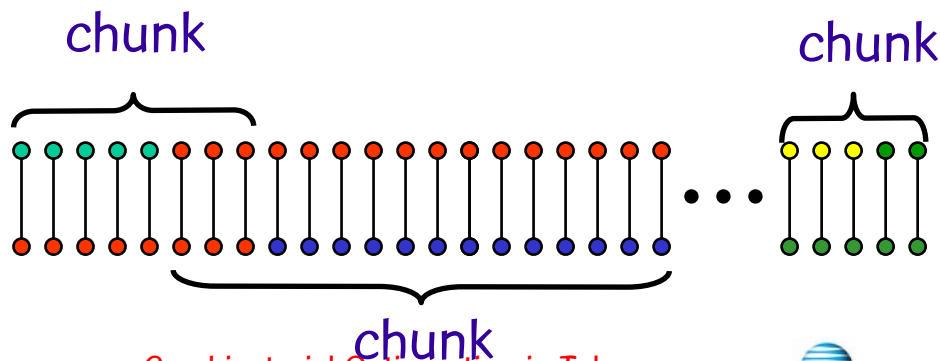
# Edge ordering

- Use levels to order edges ( $k=0,1,2,\dots$ )
  - Edges in level  $k$
  - Edges from level  $k$  to level  $k+1$



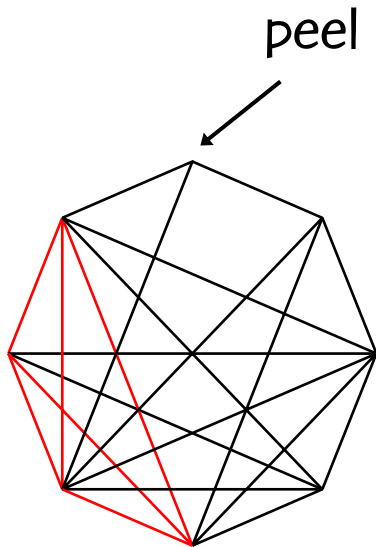
# Chunking & peeling

- Start with all edges in  $E$  (set is massive)
- Repeat
  - Create a subgraph  $G'$  with one or more chunks
  - Find large clique (of size  $c'$ ) in  $G'$
  - Peel from  $G$  all vertices  $v$  with  $\deg(v) < c'$
  - $E = E(G)$

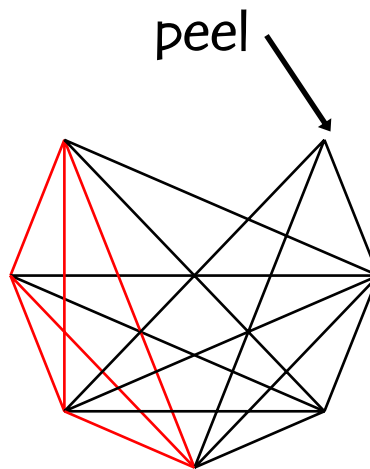


# Peeling

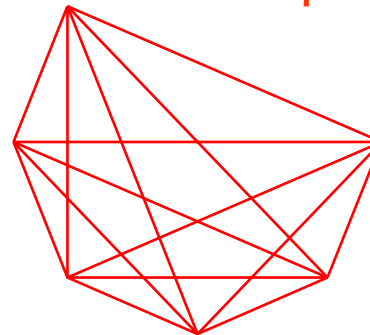
- Peeling is applied recursively



Clique of size 4

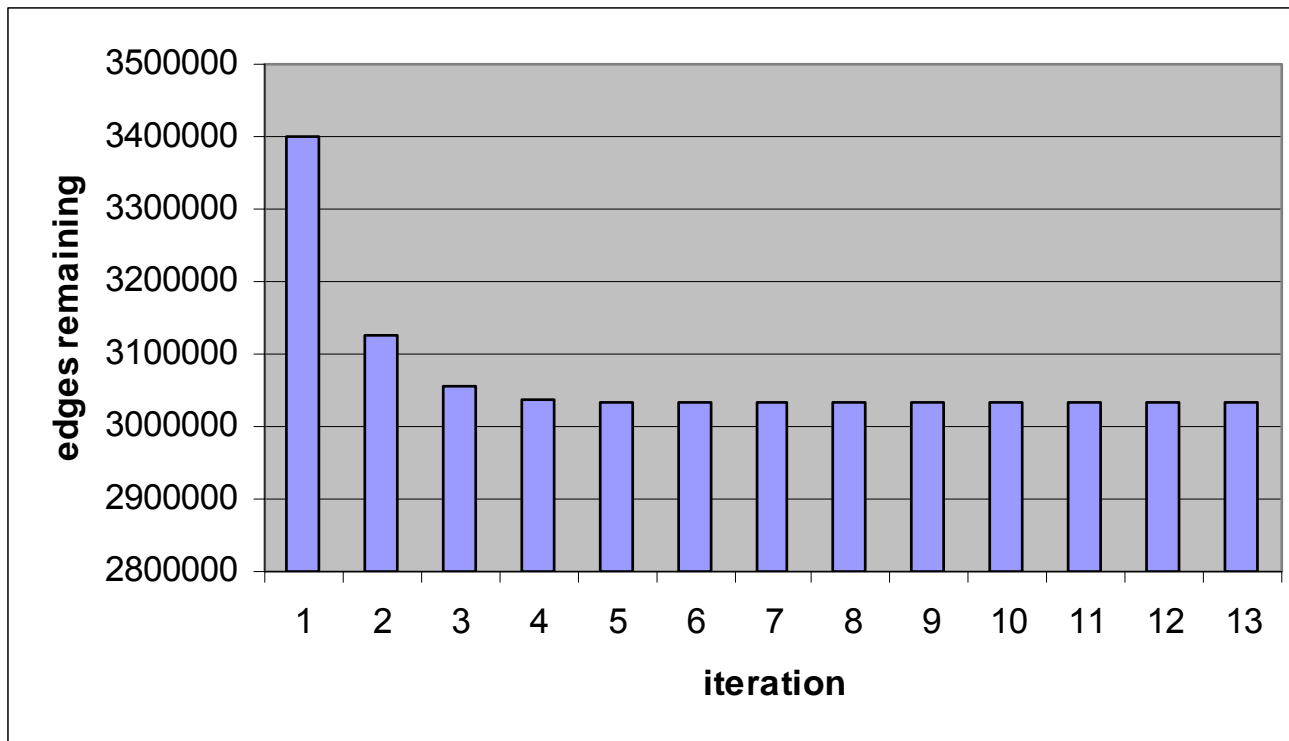


Clique of size 5



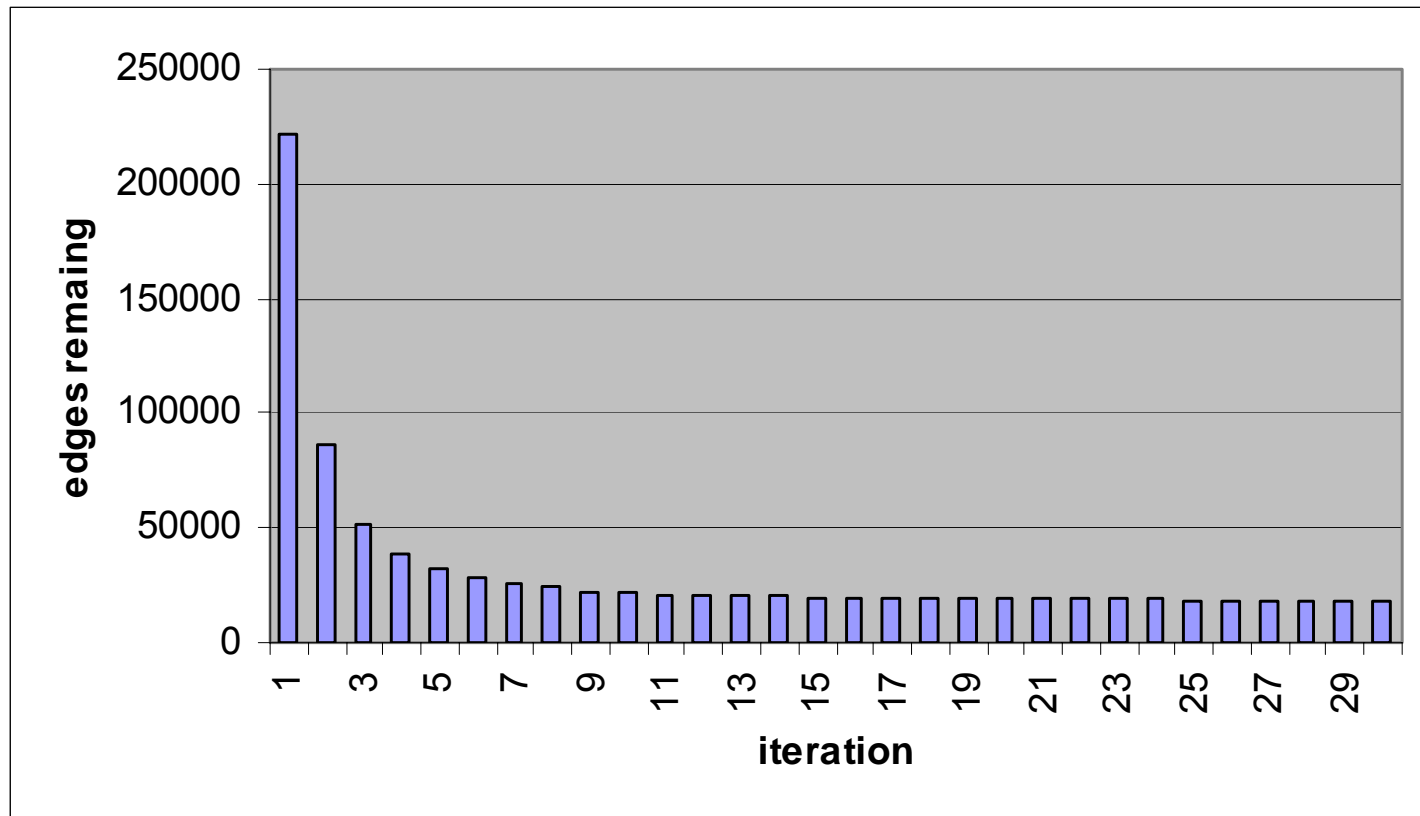
# Peeling with degree = 2

reduction from 3.4 M edges to 3.0 M edges



# Peeling with degree = 14

reduction from 3.0 M edges to 18.3 K edges

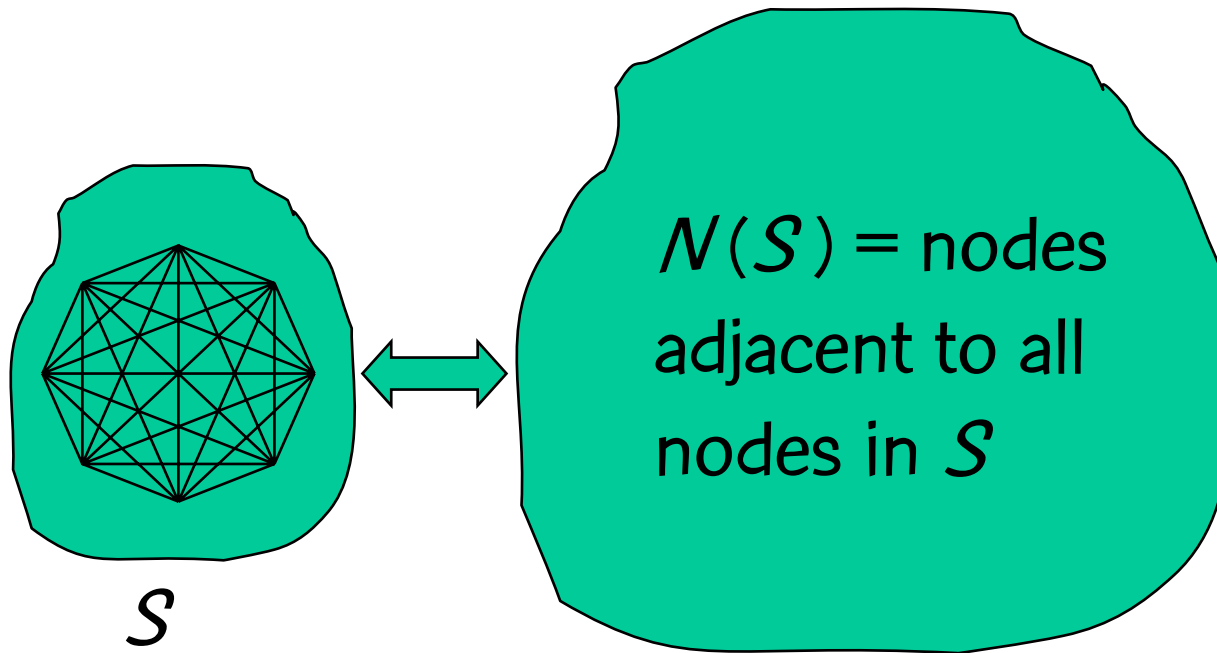


# Finding cliques

- GRASP for max clique
  - multi-start
    - construct clique using randomized greedy algorithm
    - attempt to improve clique using 2-exchange local search
    - store all cliques found in construction & local search

# Greedy vertex choice

Choose  $v \in N(S)$  with  $\max \deg_{N(S)} \{v \in N(S)\}$ .

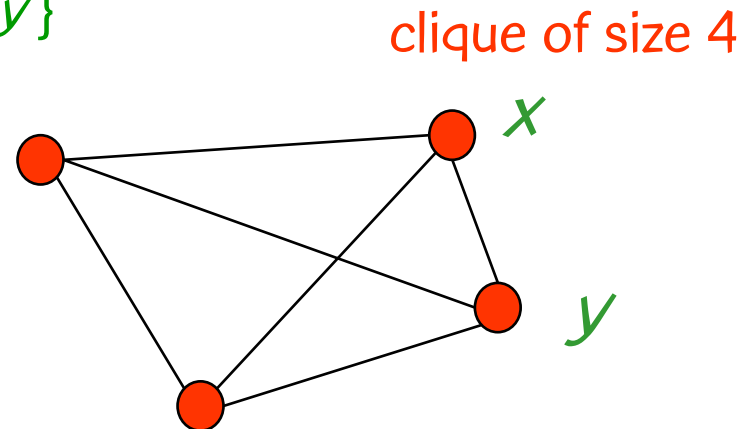
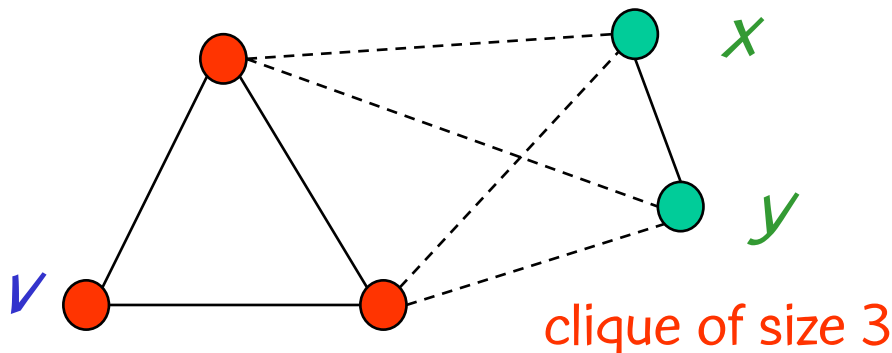




# (2,1) exchange local search

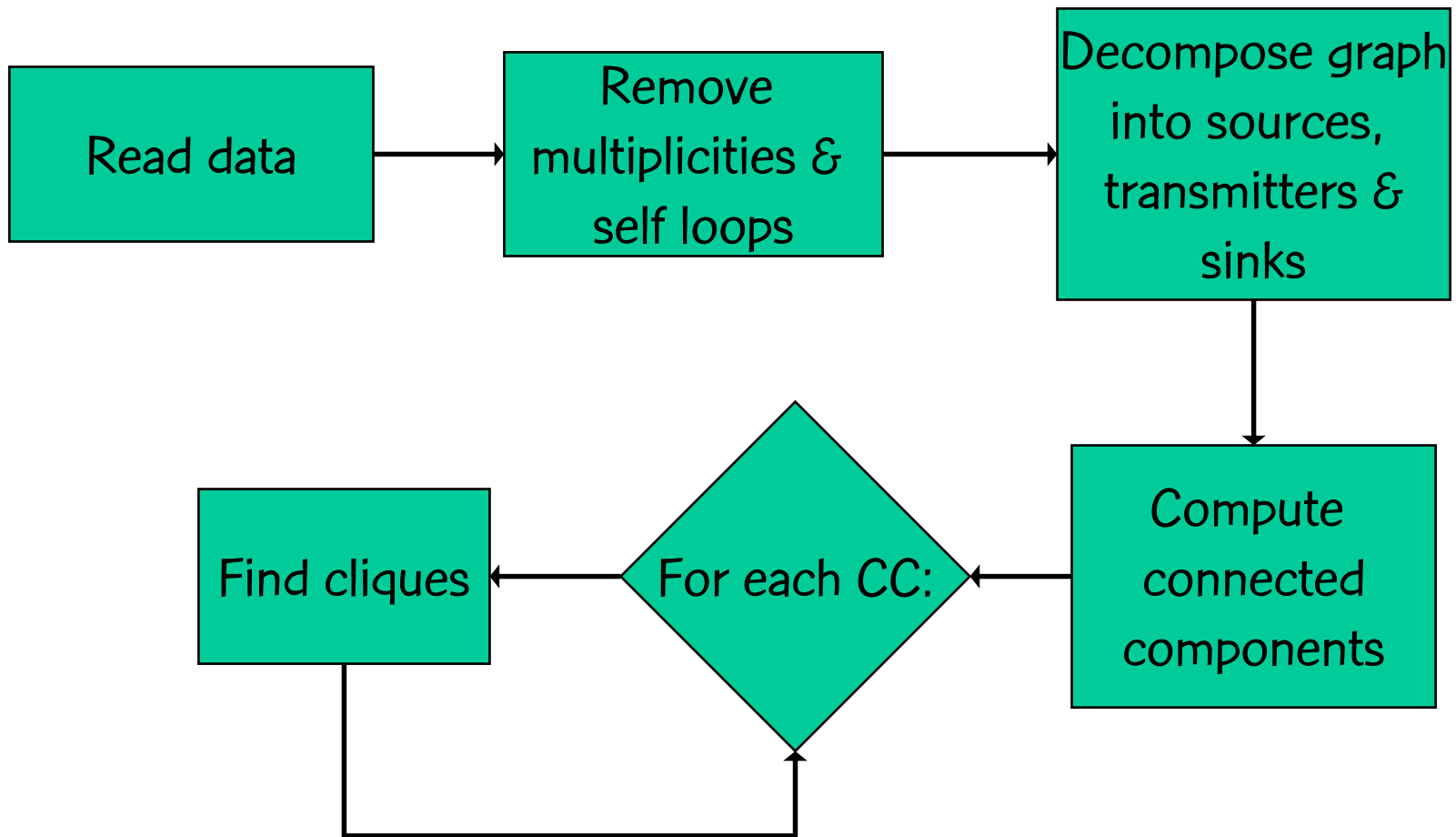
- for each vertex  $v$  in clique  $S$ 
  - while  $\exists$  an edge  $(x, y) \in E$  with  $x$  and  $y$  adjacent to all vertices in  $S \setminus \{v\}$ 
    - remove  $v$  from  $S$  and add  $x$  and  $y$  to  $S$ :

$$S = S \setminus \{v\} \cup \{x\} \cup \{y\}$$



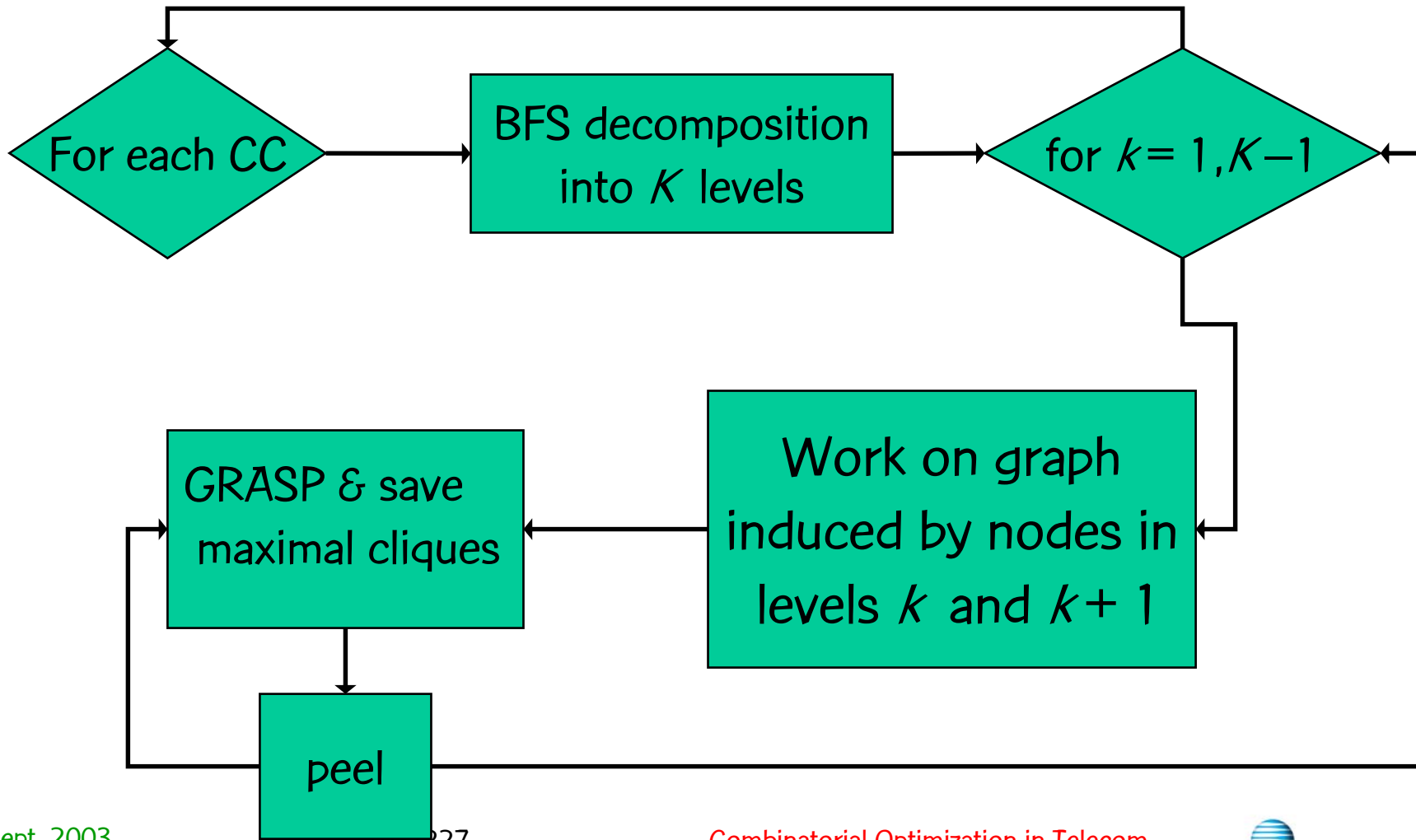
# Software platform

external & semi-external memory algorithms



# Software platform

## computing cliques



# Mining for cliques

## examples

- 12 hours of calls
  - 53M nodes, 170M edges
  - 3.6M connected components (only 302K had more than three nodes)
    - 255 self loops, 2.7M pairs, and 598K triplets
  - Giant CC has 45M nodes
  - Found cliques of size up to 30 nodes in giant CC.
  - Found quasi-cliques of size 44 (90% density), 57 (80%), 65 (70%), and 98 (50%) in giant CC.

# Application 5:

# Internet traffic engineering

# Internet traffic engineering

- Internet traffic has been doubling each year  
[Coffman & Odlyzko, 2001]
- In the 1995-96 period, there was a doubling of traffic each three months!
  - Web browsers were introduced.
- Increasingly heavy traffic (due to video, voice, etc.) will raise the requirements of the Internet of tomorrow.

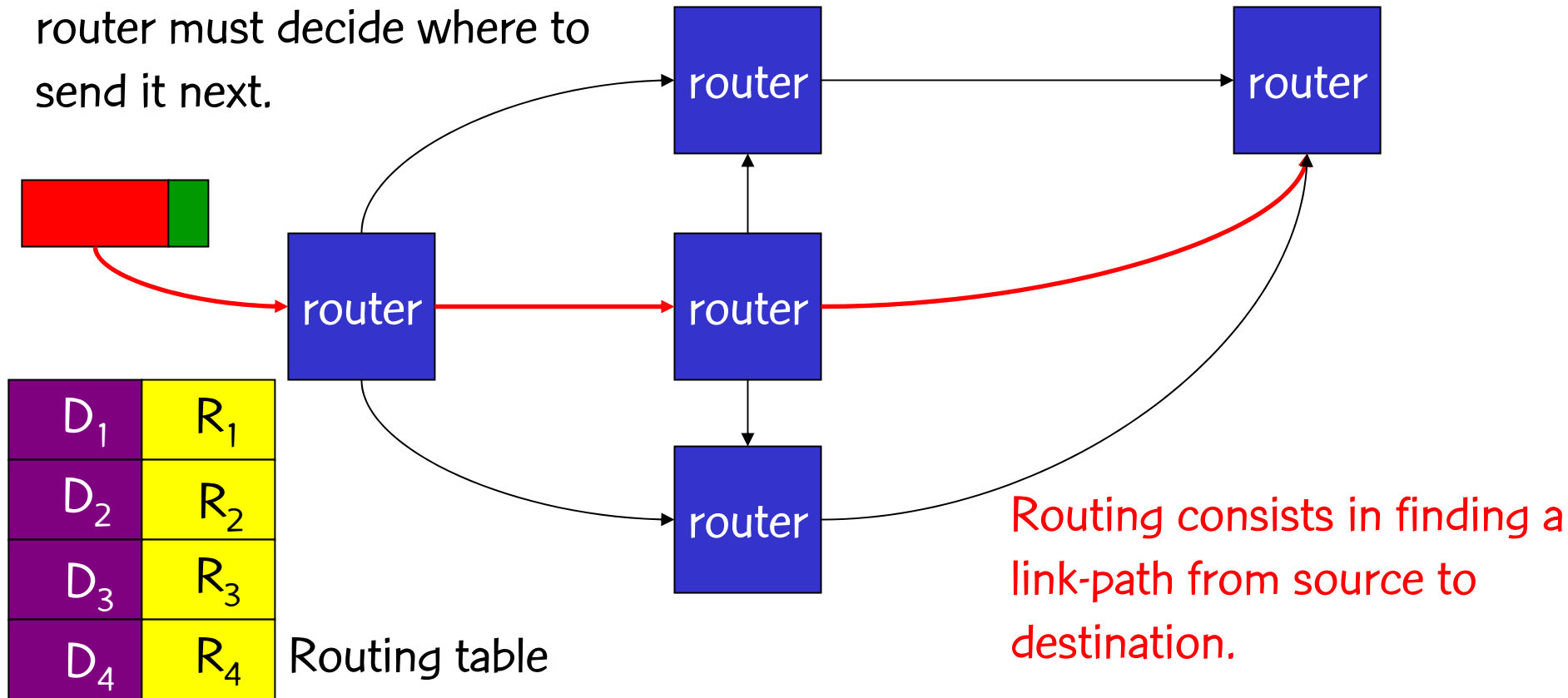
# Internet traffic engineering

- **Objective:** make more efficient use of existing network resources.
- **Routing** of traffic can have a major impact on efficiency of network resource utilization.

# Packet routing

When packet arrives at router, router must decide where to send it next.

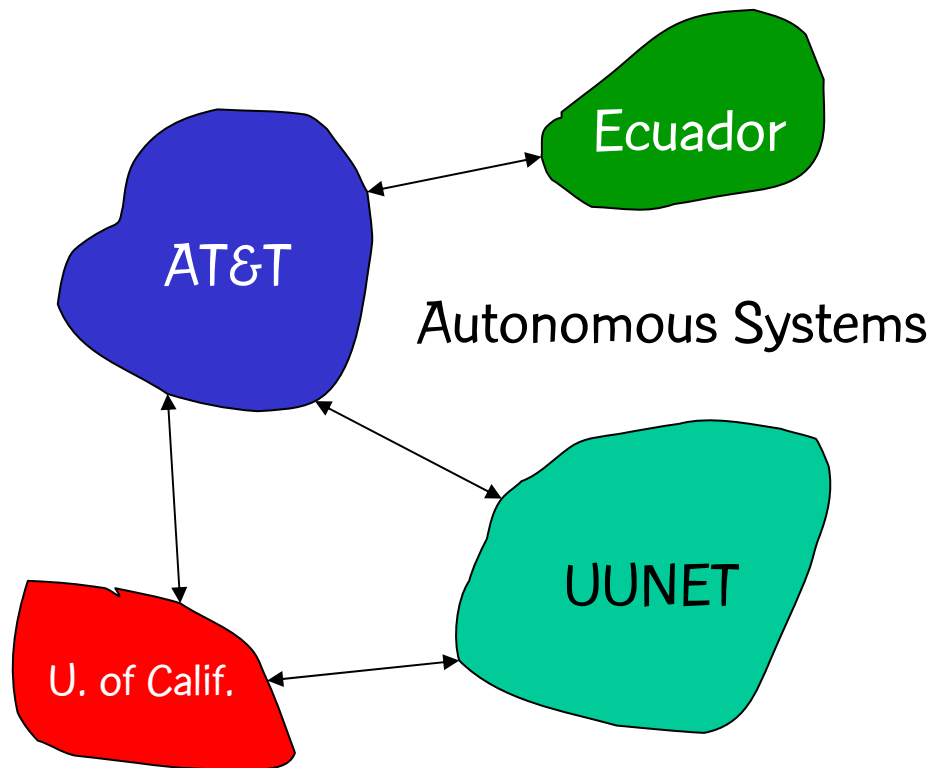
Packet's final destination.





# OSPF (Open Shortest Path First)

- **OSPF** is a commonly used intra-domain routing protocol (IGP).
- **Routers exchange routing information** with all other routers in the autonomous system (AS).
  - Complete network topology knowledge is available to all routers, i.e. state of all routers and links in the AS.



# OSPF routing

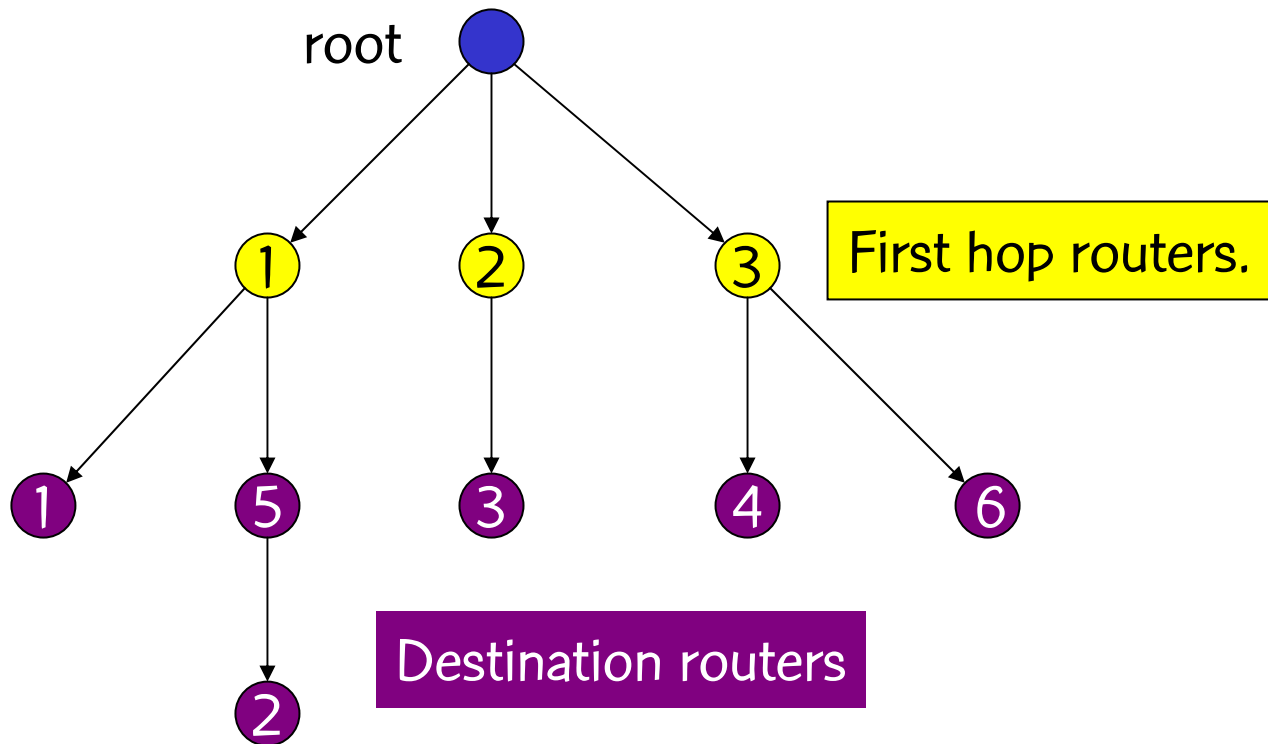
- Assign an integer weight  $\in [1, w_{max}]$  to each link in AS. In general,  $w_{max} = 65535 = 2^{16} - 1$ .
- Each router computes tree of shortest weight paths to all other routers in the AS, with itself as the root, using Dijkstra's algorithm.

# OSPF routing

Routing table

$D_1$	$R_1$
$D_2$	$R_1$
$D_3$	$R_2$
$D_4$	$R_3$
$D_5$	$R_1$
$D_6$	$R_3$

Routing table is filled with first hop routers for each possible destination.

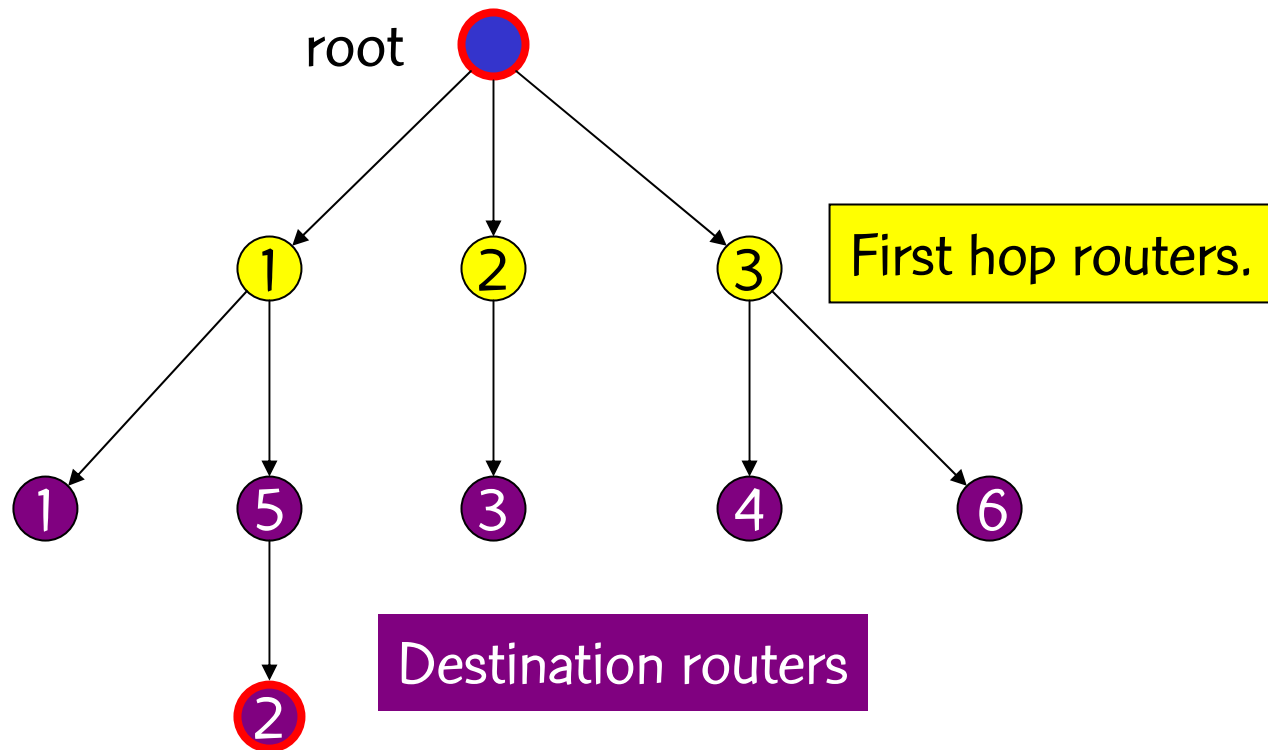


# OSPF routing

Routing table

$D_1$	$R_1$
$D_2$	$R_1$
$D_3$	$R_2$
$D_4$	$R_3$
$D_5$	$R_1$
$D_6$	$R_3$

Routing table is filled with first hop routers for each possible destination.

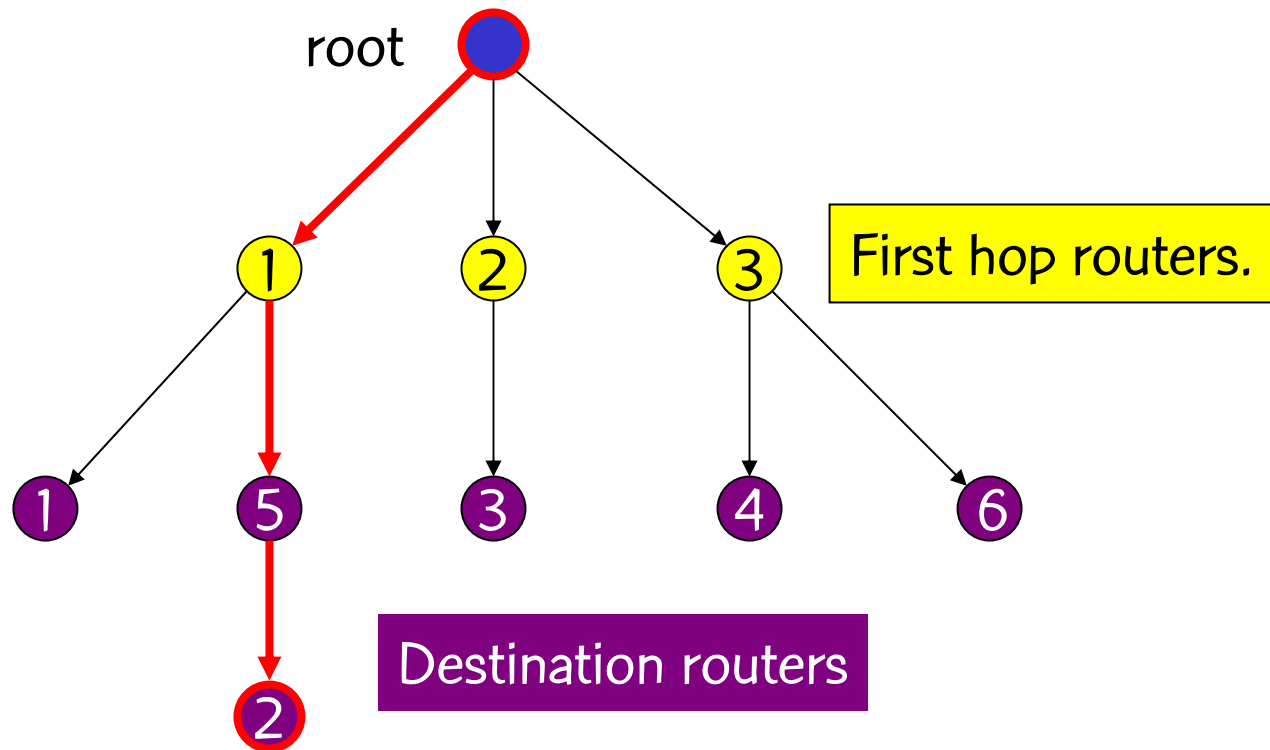


# OSPF routing

Routing table

$D_1$	$R_1$
$D_2$	$R_1$
$D_3$	$R_2$
$D_4$	$R_3$
$D_5$	$R_1$
$D_6$	$R_3$

Routing table is filled with first hop routers for each possible destination.

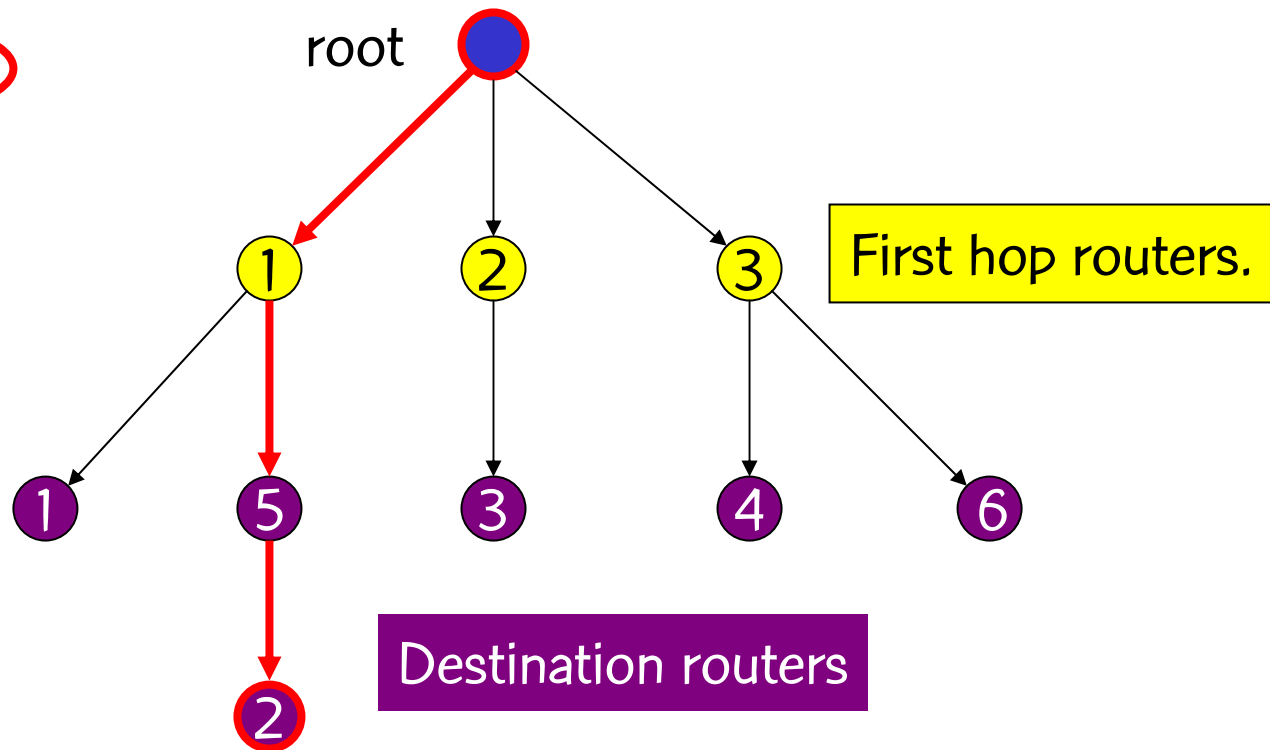


# OSPF routing

Routing table

$D_1$	$R_1$
$D_2$	$R_1$
$D_3$	$R_2$
$D_4$	$R_3$
$D_5$	$R_1$
$D_6$	$R_3$

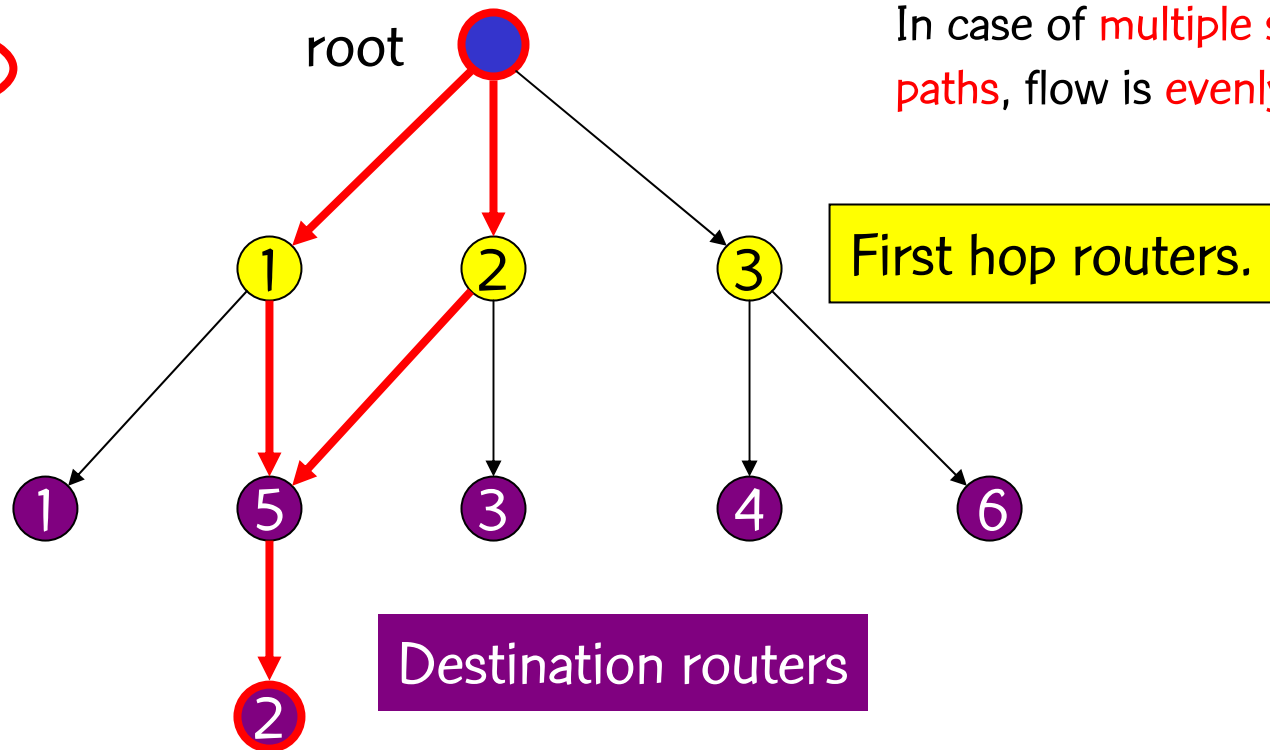
Routing table is filled with first hop routers for each possible destination.



# OSPF routing

Routing table

$D_1$	$R_1$
$D_2$	$R_1, R_2$
$D_3$	$R_2$
$D_4$	$R_3$
$D_5$	$R_1$
$D_6$	$R_3$



Routing table is filled with first hop routers for each possible destination. In case of **multiple shortest paths**, flow is **evenly split**.

# OSPF weight setting

- OSPF weights are assigned by network operator.
  - CISCO assigns, by default, a weight proportional to the inverse of the link bandwidth (Inv Cap).
  - If all weights are unit, the weight of a path is the number of hops in the path.
- We propose a hybrid genetic algorithm to find good OSPF weights.
  - Memetic algorithm
  - Genetic algorithm with optimized crossover



# Minimization of congestion

- Consider the directed capacitated network  $G = (N, A, c)$ , where  $N$  are routers,  $A$  are links, and  $c_a$  is the capacity of link  $a \in A$ .
- We use the measure of Fortz & Thorup (2000) to compute congestion:

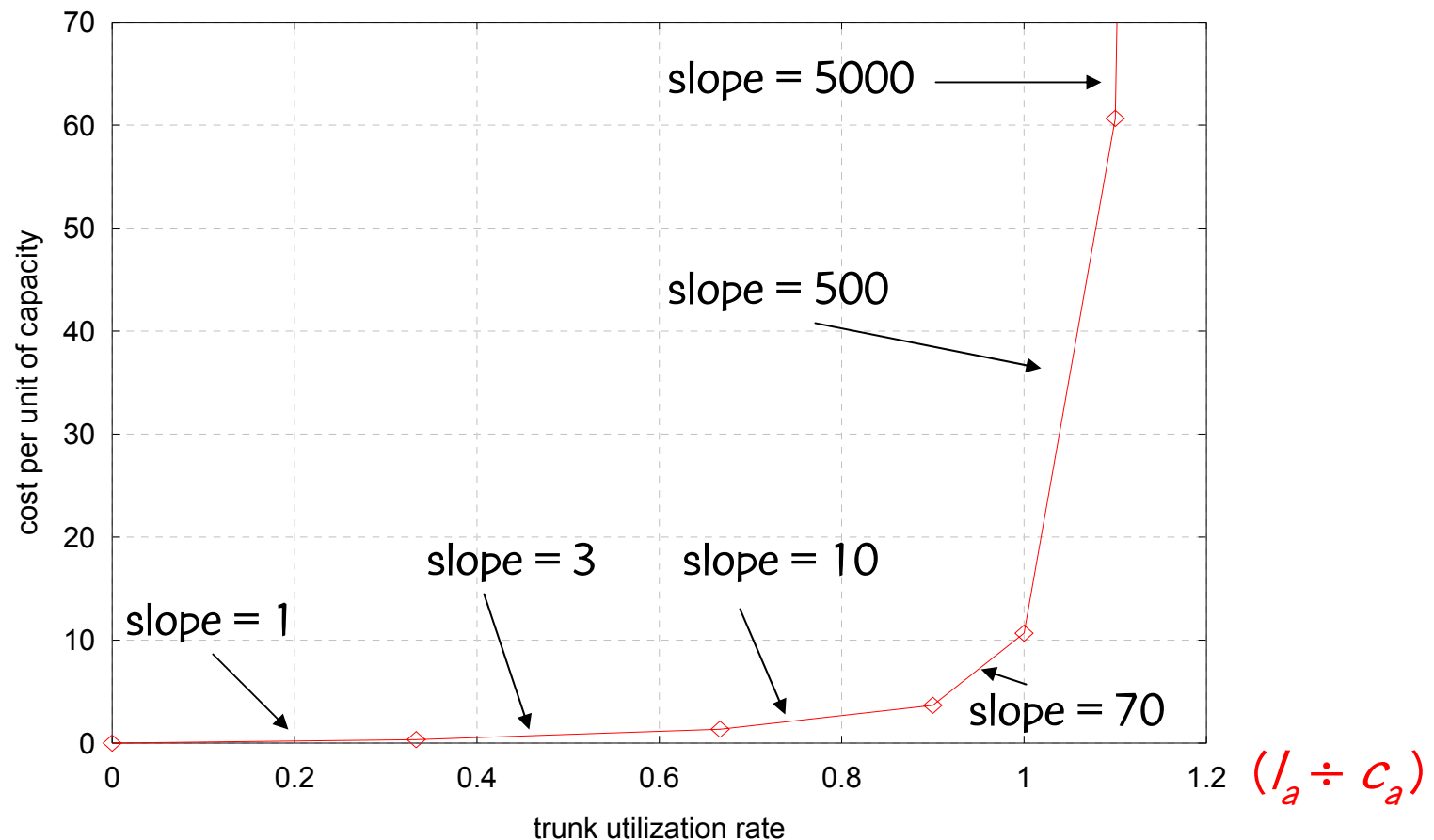
$$\Phi = \Phi_1(l_1) + \Phi_2(l_2) + \dots + \Phi_{|A|}(l_{|A|})$$

where  $l_a$  is the load on link  $a \in A$ ,

$\Phi_a(l_a)$  is piecewise linear and convex,

$\Phi_a(0) = 0$ , for all  $a \in A$ .

# Piecewise linear and convex $\Phi_a(I_a)$ link congestion measure



# OSPF weight setting problem

- Given a directed network  $G = (N, A)$  with link capacities  $c_a \in A$  and demand matrix  $D = (d_{s,t})$  specifying a demand to be sent from node  $s$  to node  $t$ :
  - Assign weights  $w_a \in [1, w_{max}]$  to each link  $a \in A$ , such that the objective function  $\Phi$  is minimized when demand is routed according to the OSPF protocol.

# Cost normalization

Consider the demand matrix  $D = (d_{s,t})$  and let  $h_{s,t}$  be the min hop count between  $s$  and  $t$ .

$$\text{Normalize } \Phi \text{ by } \Phi_{uncap} = \sum_{(s,t) \in N \times N} d_{s,t} h_{s,t}$$

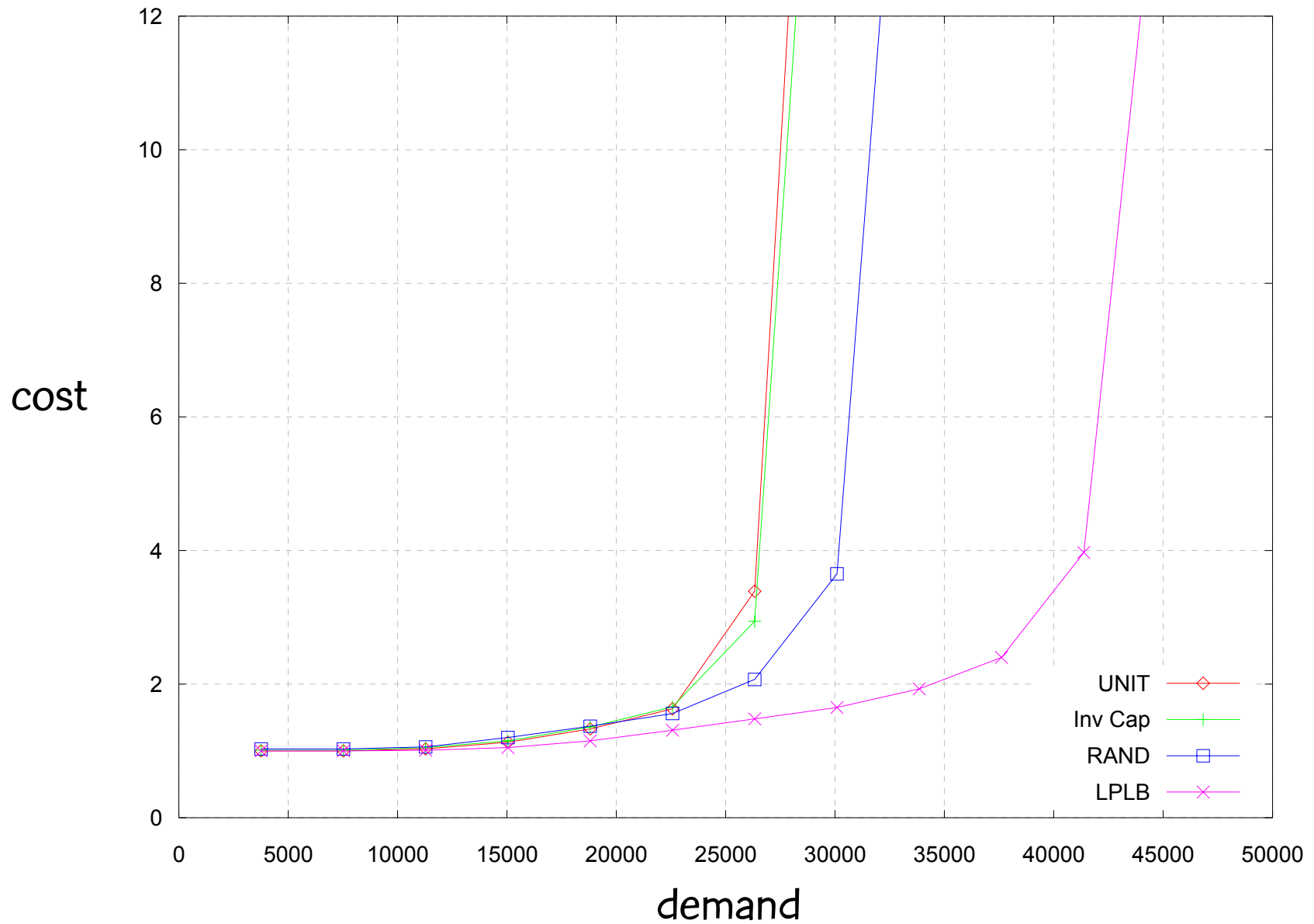
Total load if all traffic goes along unit weight shortest paths.

$$\text{Normalized cost: } \Phi^* = \Phi / \Phi_{uncap}$$

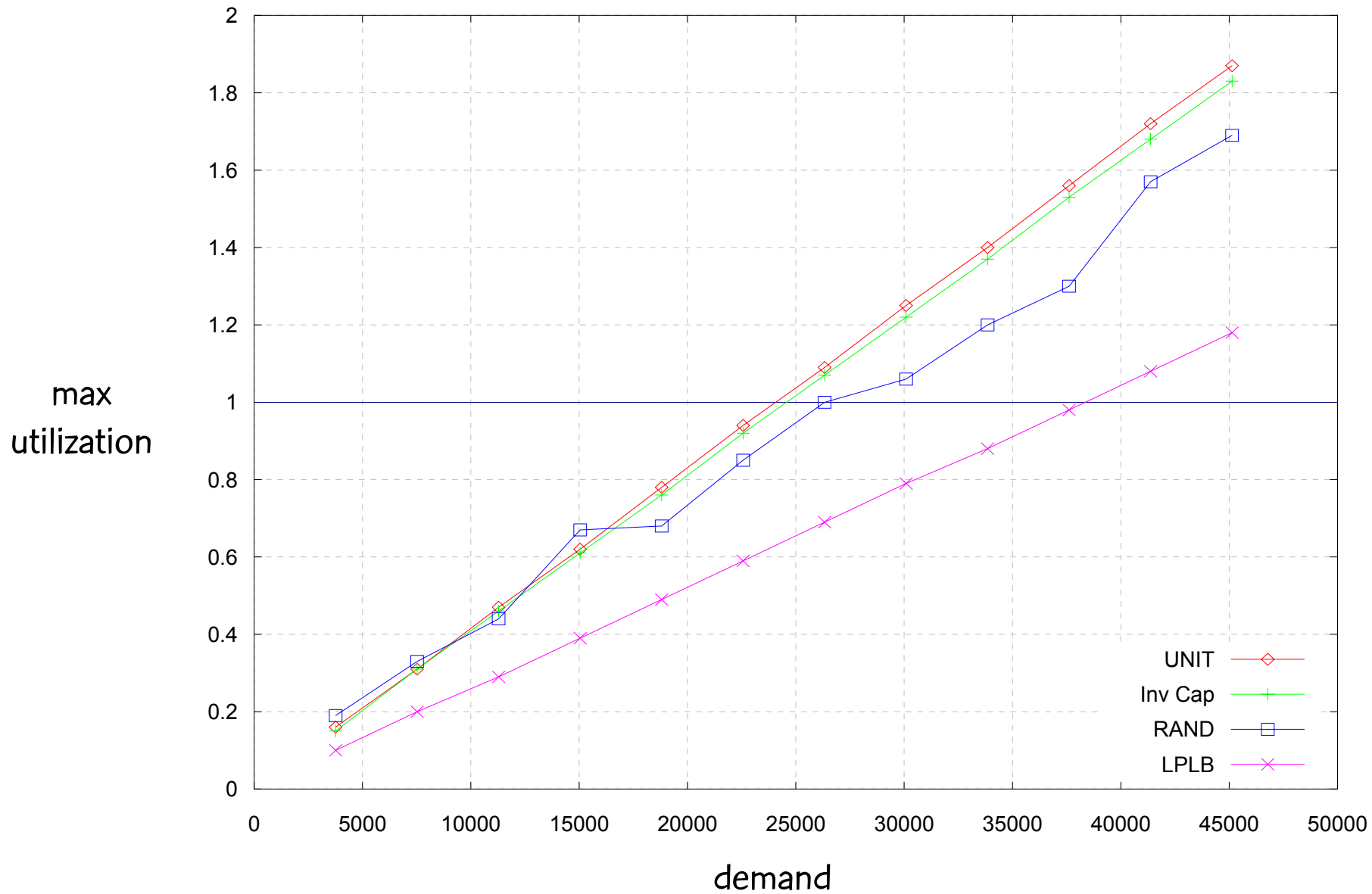
Normalized cost  $\Phi^* = \Phi / \Phi_{uncap}$

- Fortz & Thorup (2000) show that:
- $1 \leq \Phi_{opt}^* \leq \Phi_{optOSPF}^* \leq \Phi_{unitOSPF}^* < 5000$
- If  $\Phi^* = 1$ , then all loads are below 1/3 of capacity.
- If a packet follows a shortest path and if all arcs are exactly full, then  $\Phi^* = 10\frac{2}{3}$
- Routing congests the network if  $\Phi^* \geq 10\frac{2}{3}$

# AT&T Worldnet backbone network (90 routers, 274 links)



# AT&T Worldnet backbone network (90 routers, 274 links)

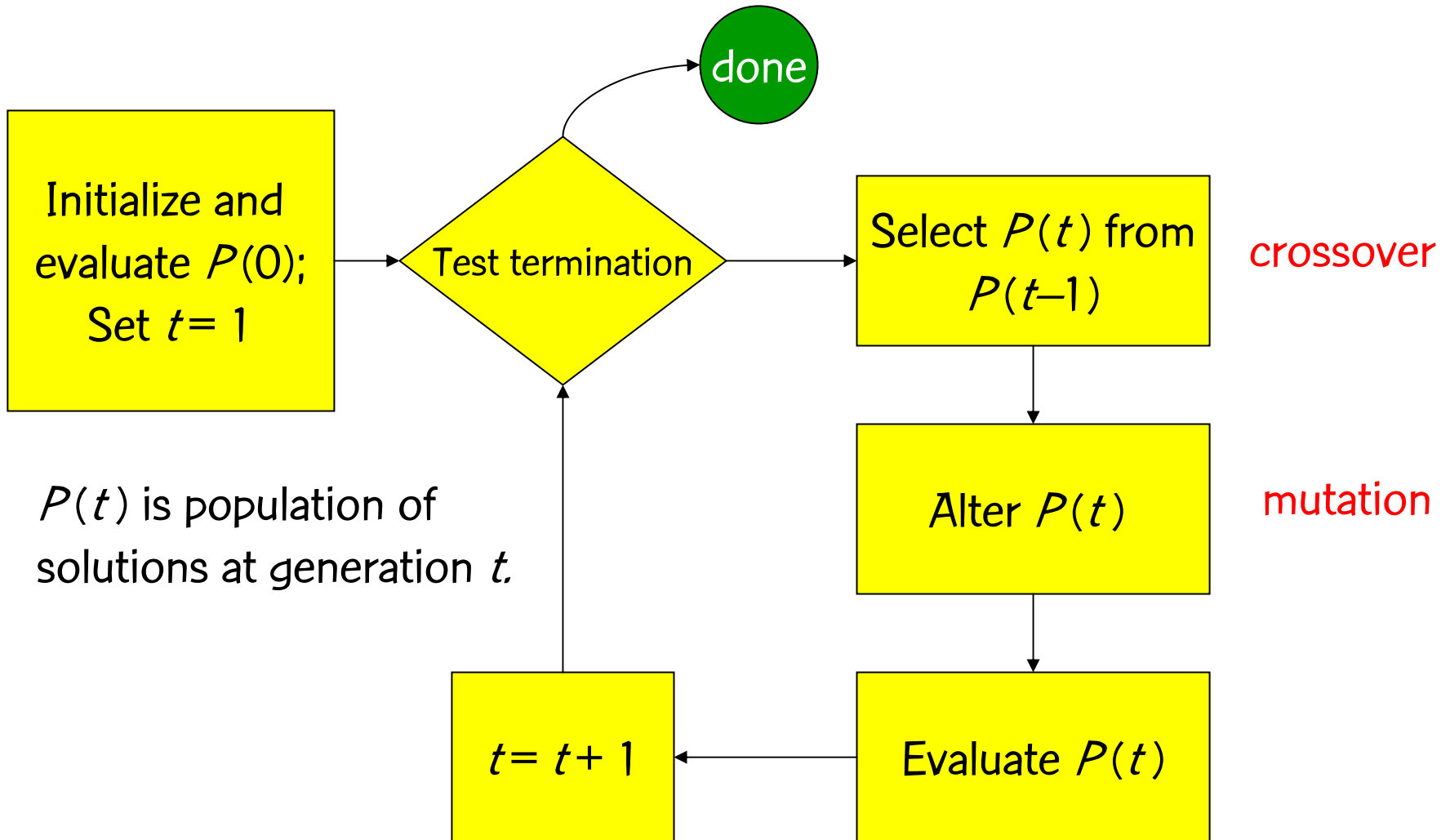


# Genetic and memetic algorithms for OSPF weight setting problem

- Genetic
  - Ericsson, Resende, & Pardalos (2002)
- Memetic
  - Buriol, Resende, Ribeiro, & Thorup (2003)



# Genetic algorithms



# Solution encoding

- A population consists of  $nPop = 50$  integer weight arrays:  $w = (w_1, w_2, \dots, w_{|A|})$ ,  
where  $w_a \in [1, w_{max} = 20]$
- All possible weight arrays correspond to feasible solutions.

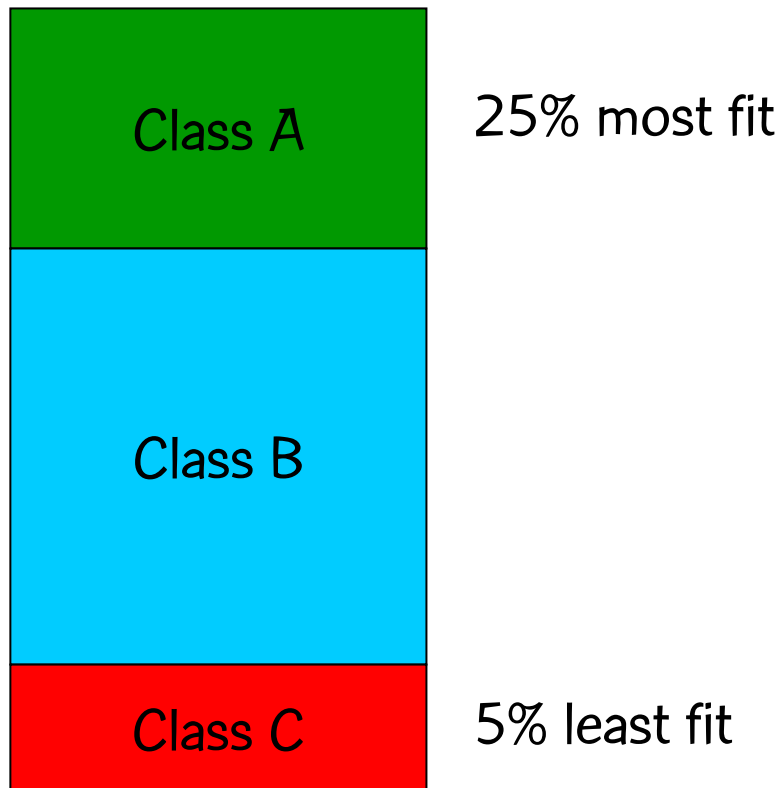
# Initial population

- $nPop$  solutions, with each weight randomly generated, uniformly in the interval  $[1, w_{max}/3]$ .

# Solution evaluation

- For each demand pair  $(s,t)$ , route using OSPF, computing demand pair loads  $l_a^{s,t}$  on each link  $a \in A$ .
- Add up demand pair loads on each link  $a \in A$ , yielding total load  $l_a$  on link.
- Compute link congestion cost  $\Phi_a(l_a)$  for each link  $a \in A$ .
- Add up costs:  $\Phi = \Phi_1(l_1) + \Phi_2(l_2) + \dots + \Phi_{|A|}(l_{|A|})$

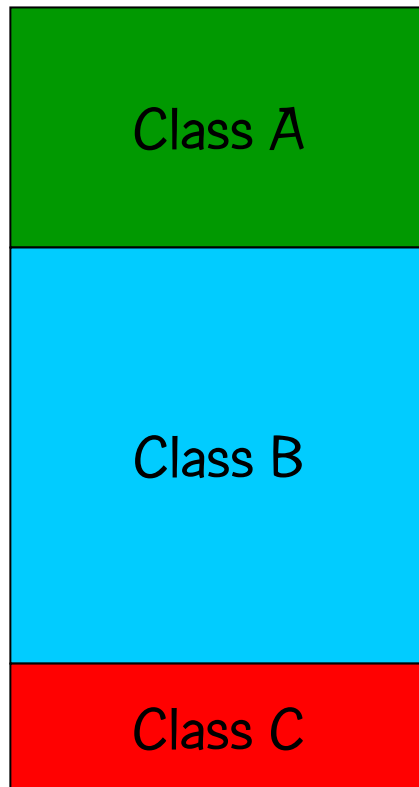
# Population partitioning



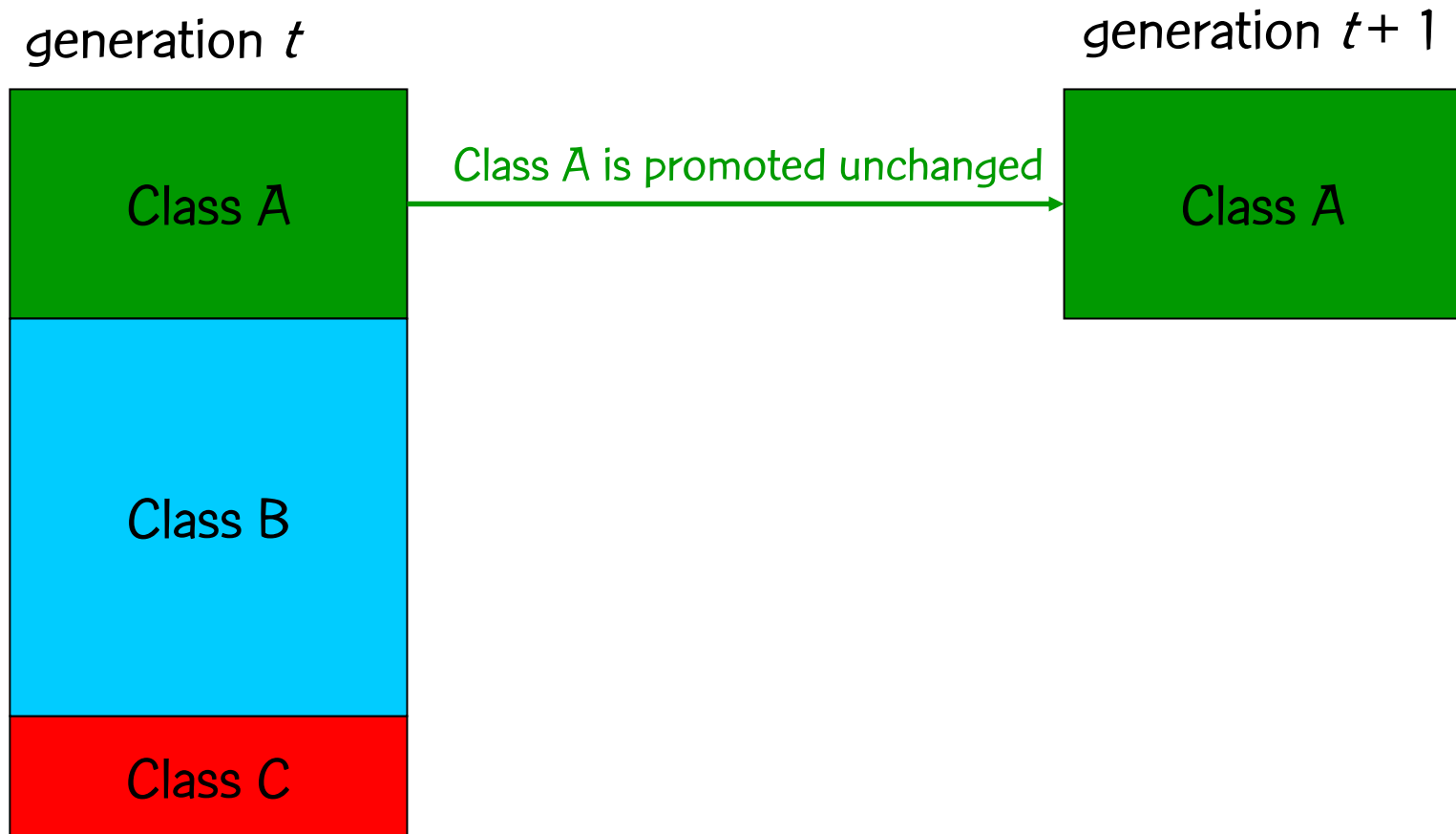
Population is sorted according to solution value  $\Phi$  and solutions are classified into three categories.

# Population dynamics

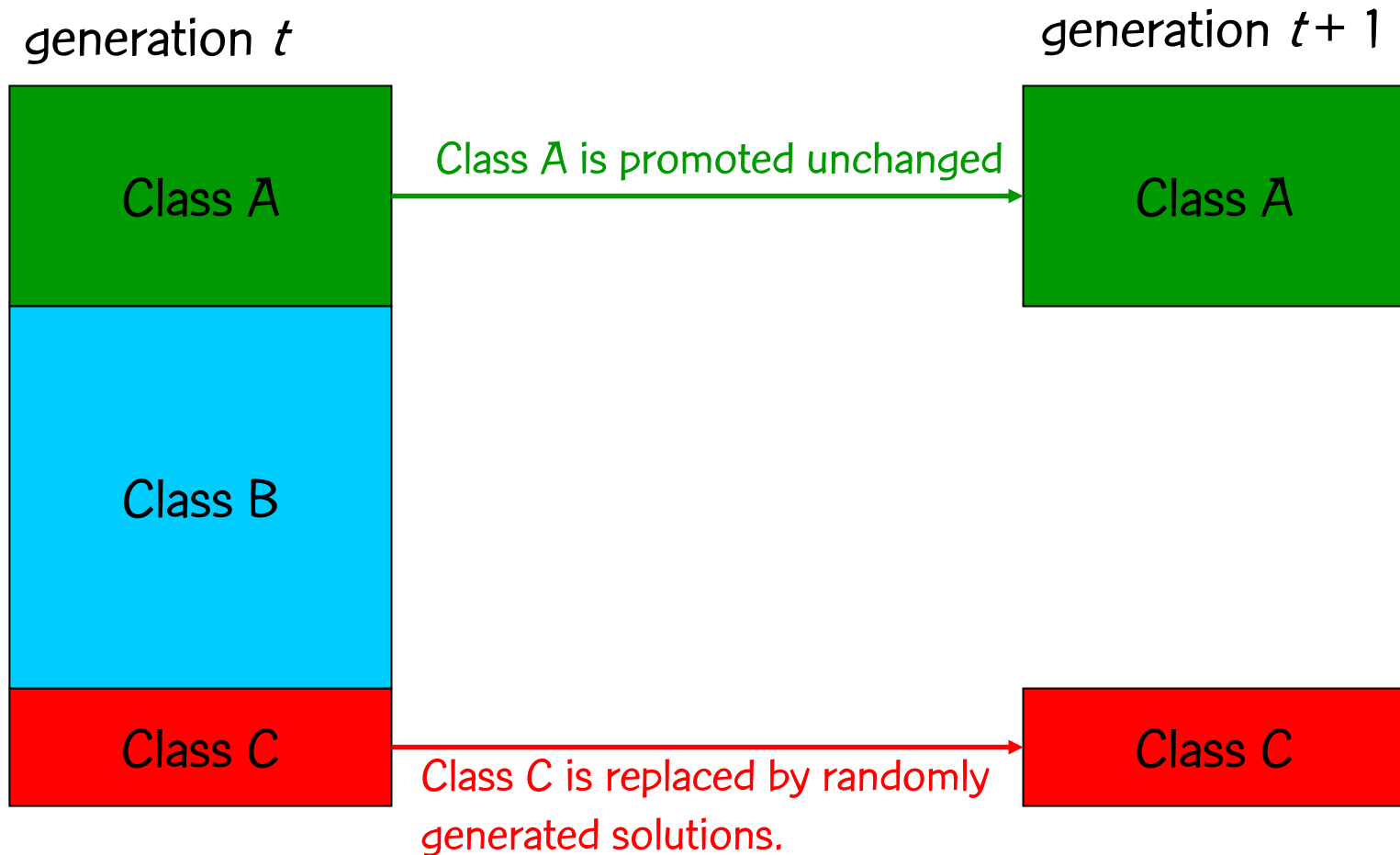
generation  $t$



# Population dynamics

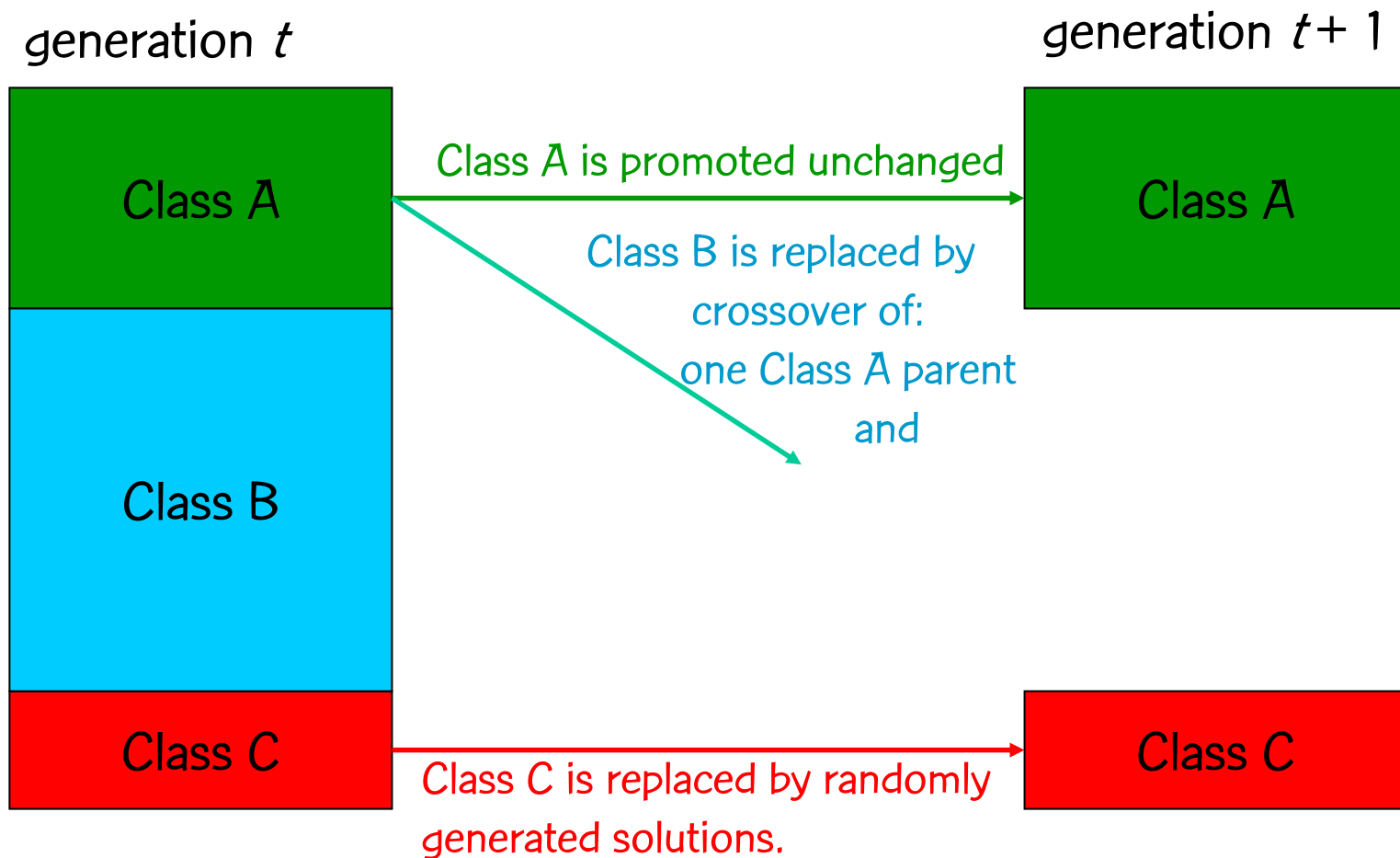


# Population dynamics

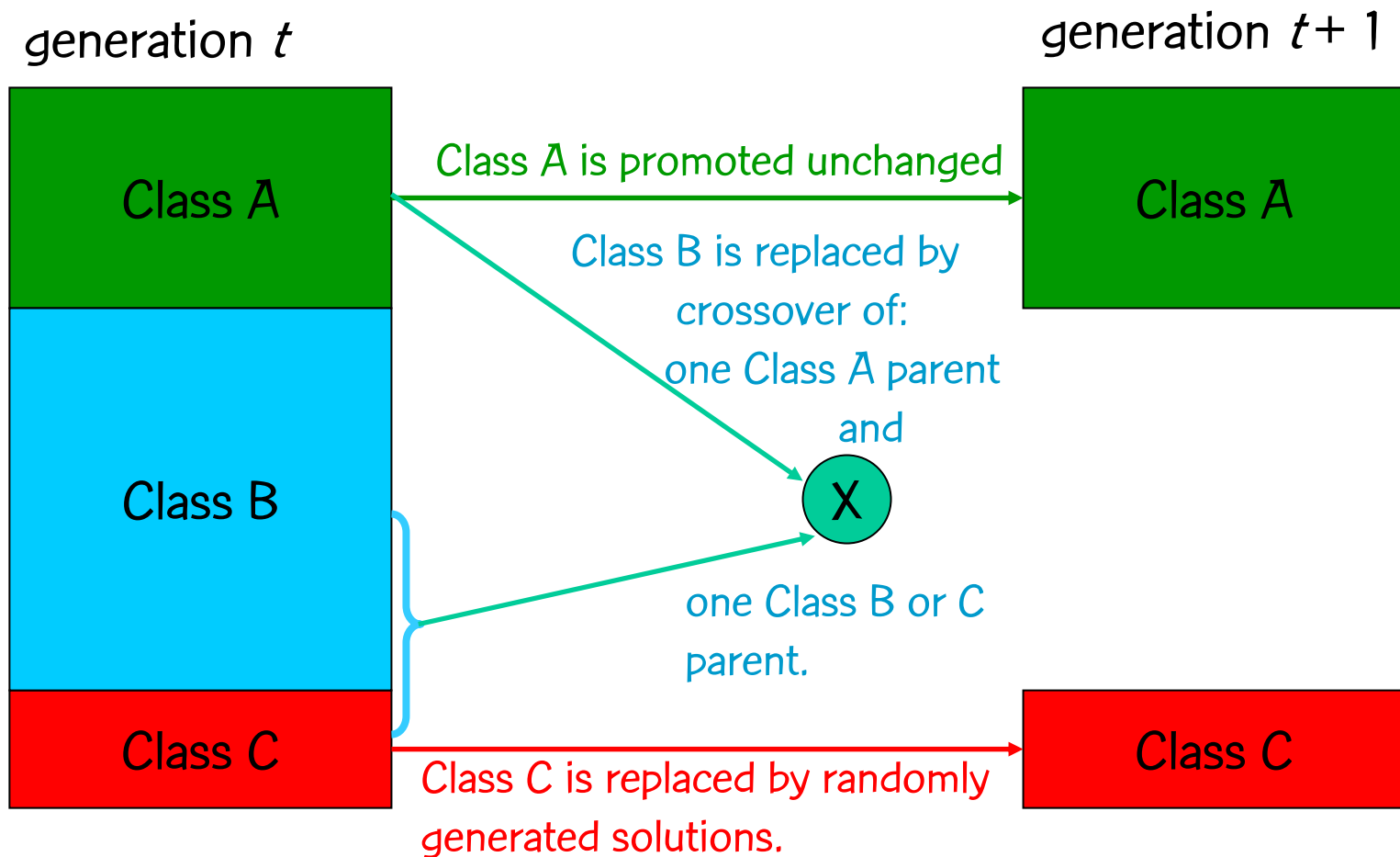




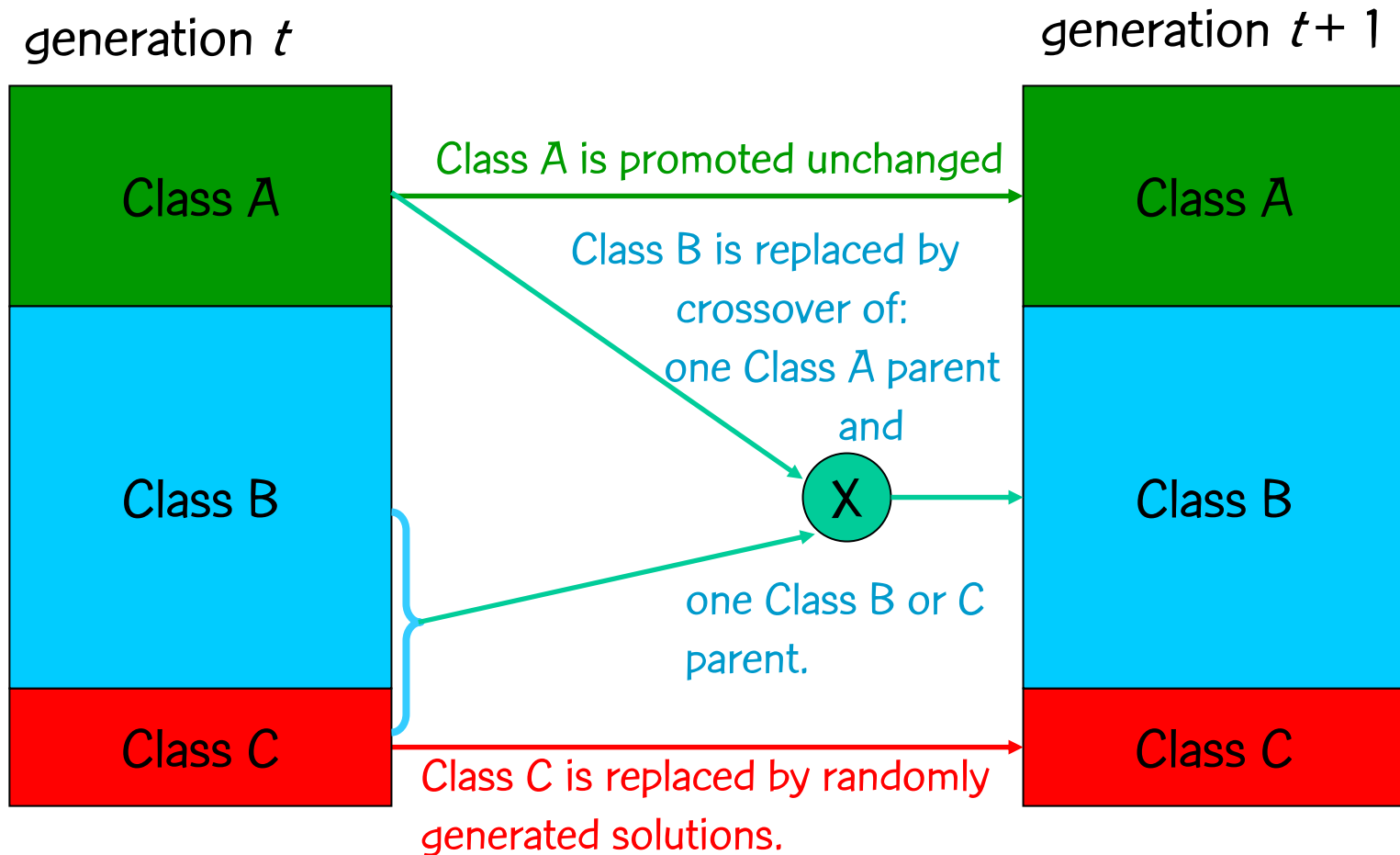
# Population dynamics



# Population dynamics



# Population dynamics



# Parent selection

- Parents are chosen at random:
  - one parent from Class A (elite).
  - one parent from Class B or C (non-elite).
- Reselection is allowed, i.e. parents can breed more than once per generation.
- Better individuals are more likely to reproduce.

# Crossover with random keys

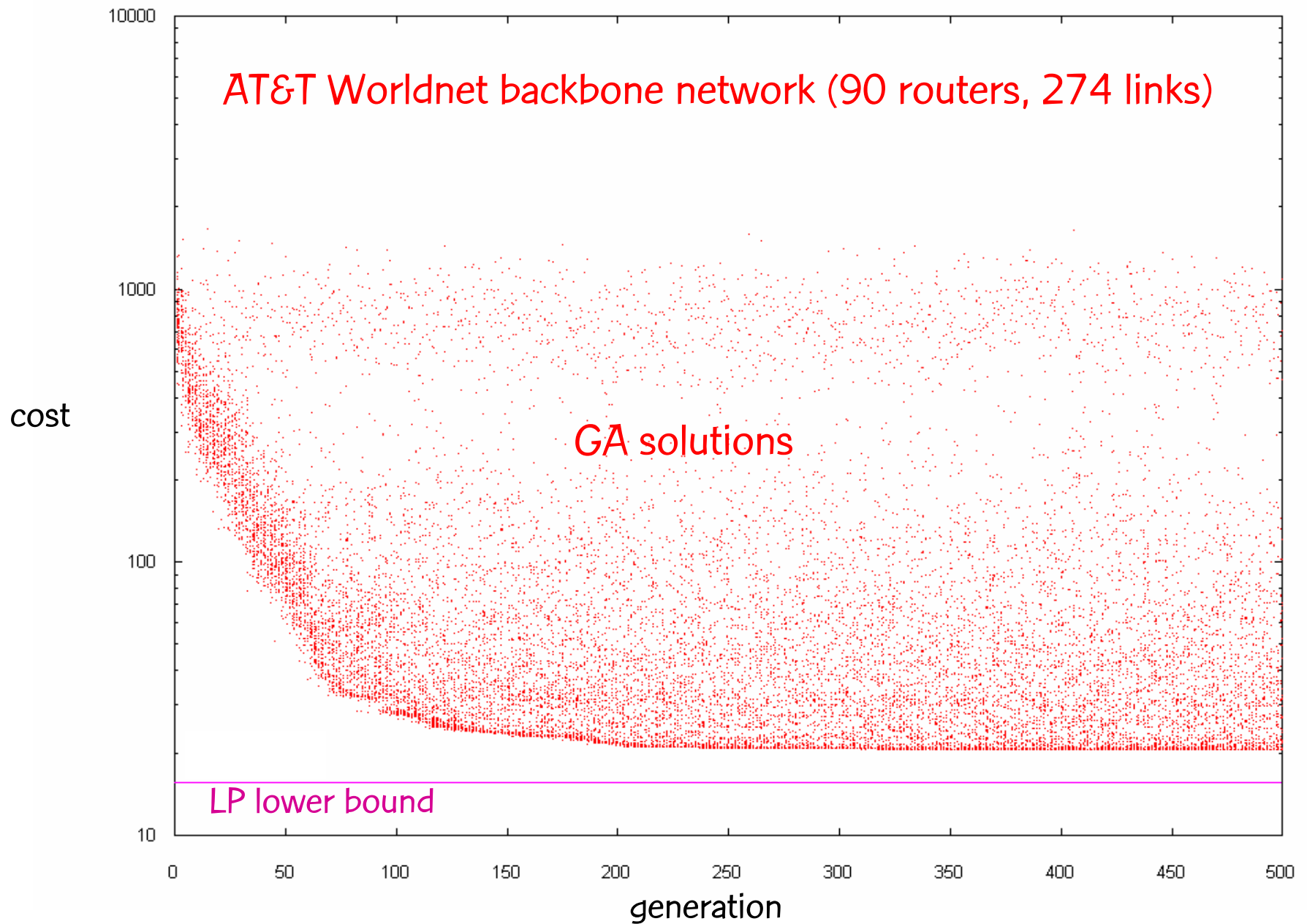
Bean (1994)

Crossover combines elite parent  $p_1$  with non-elite parent  $p_2$  to produce child  $c$ :

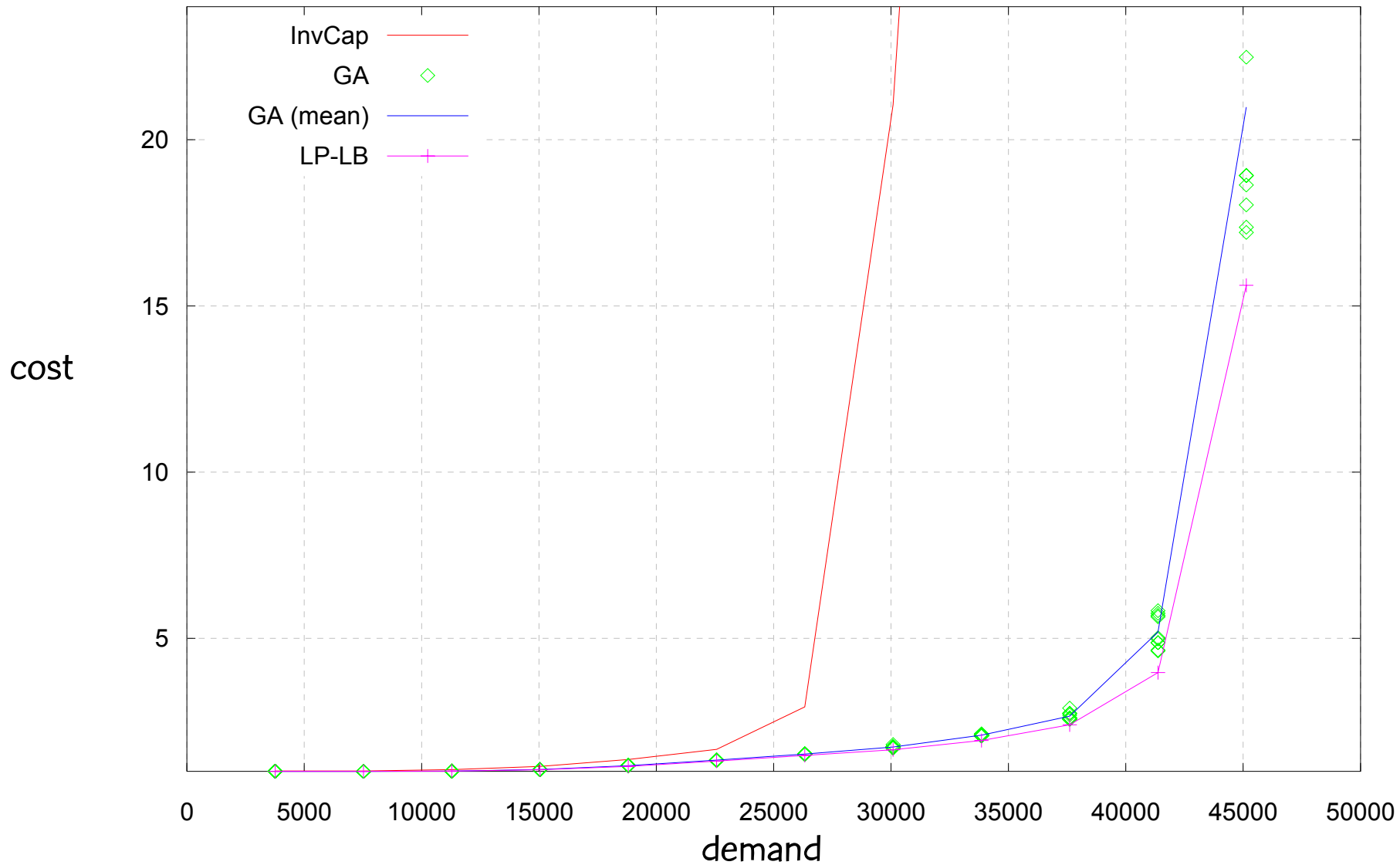
With small probability child has single gene mutation.

Child is more likely to inherit gene of elite parent.

```
for all genes  $i = 1, 2, \dots, |A|$  do
  if  $\text{rrandom}[0,1] < 0.01$  then
     $c[i] = \text{irandom}[1, w_{\max}]$ 
  else if  $\text{rrandom}[0,1] < 0.7$  then
     $c[i] = p_1[i]$ 
  else  $c[i] = p_2[i]$ 
end
```

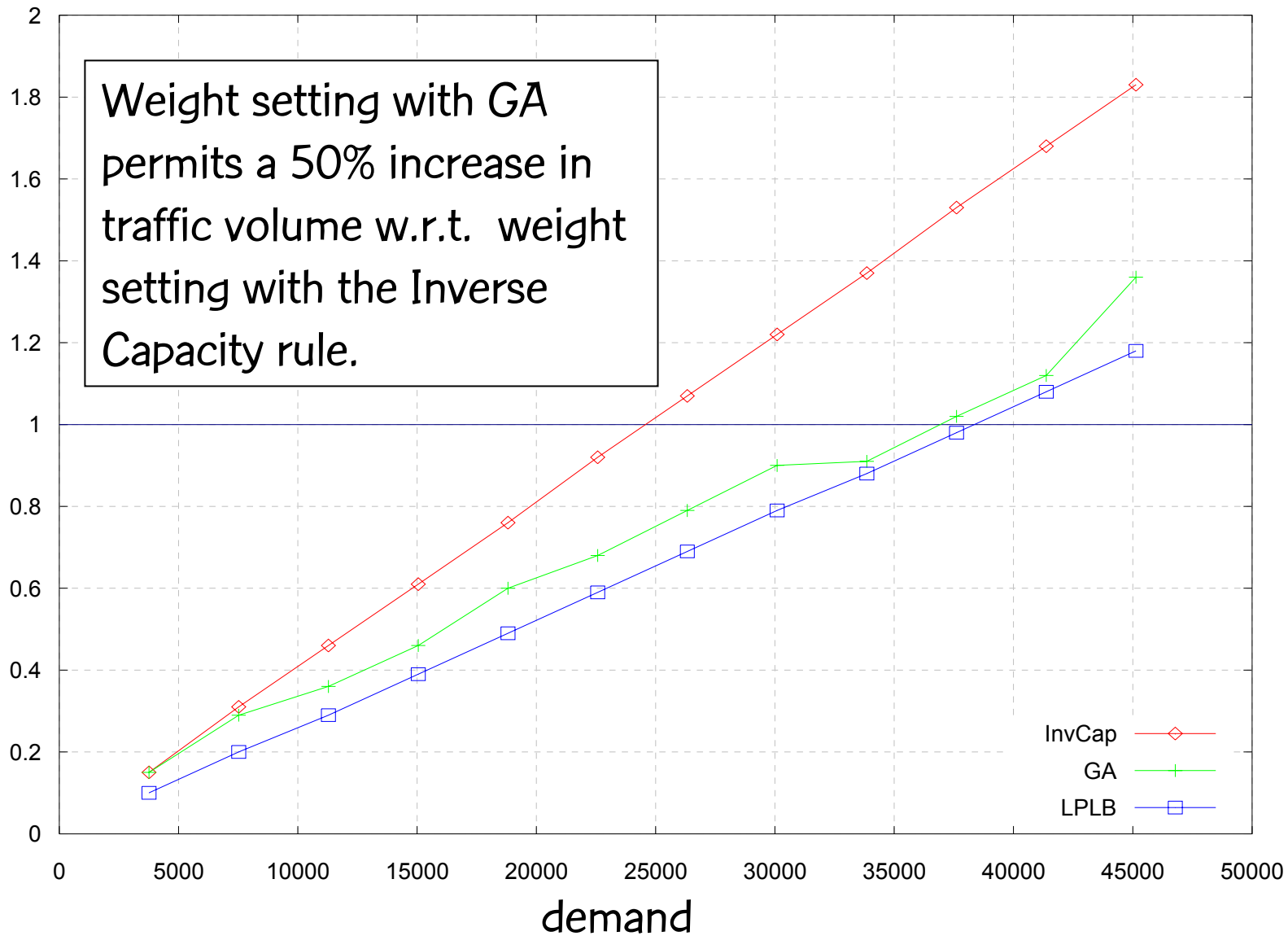


# AT&T Worldnet backbone network (90 routers, 274 links)



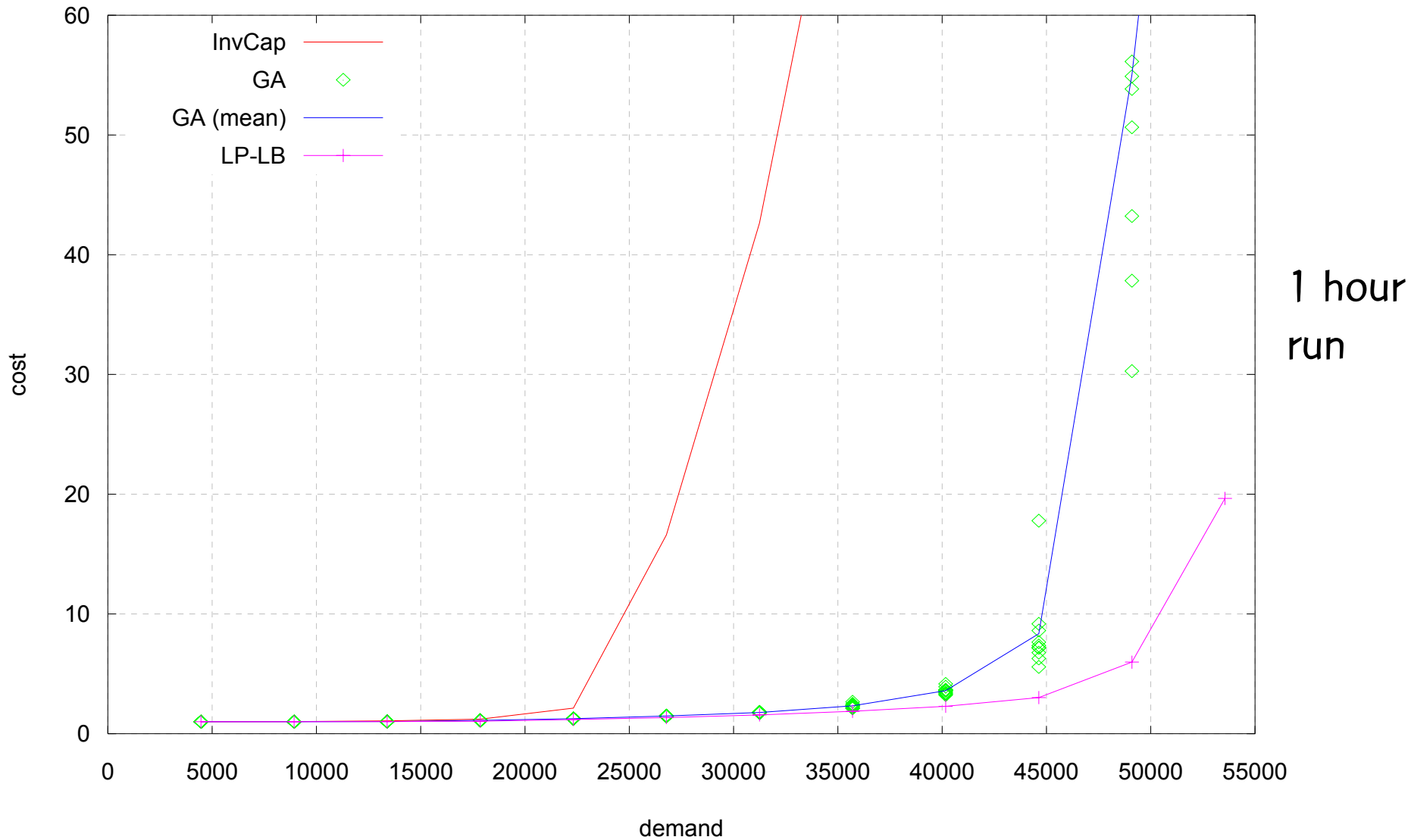
# AT&T Worldnet backbone network (90 routers, 274 links)

max  
utilization

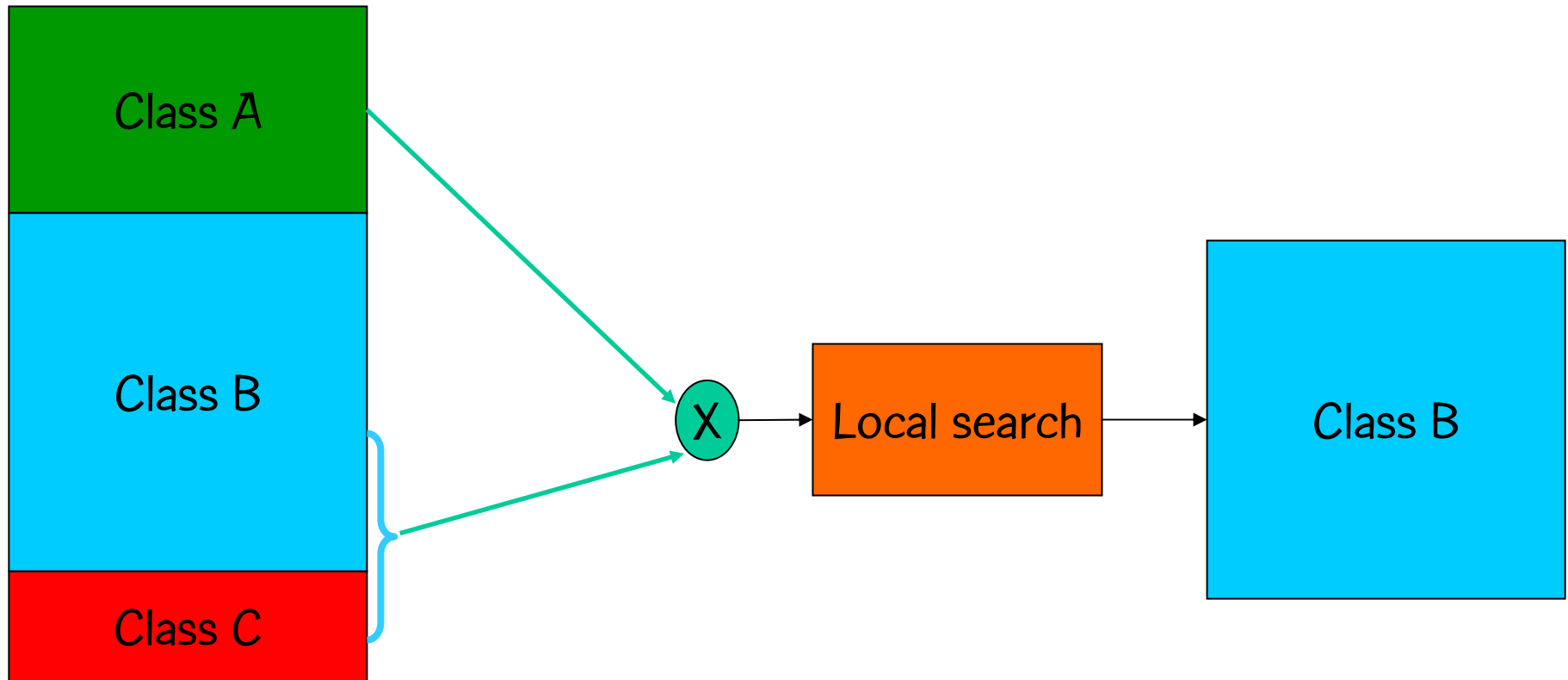




## Rand50a: random graph with 50 nodes and 245 arcs.



# Optimized crossover = crossover + local search



# Fast local search

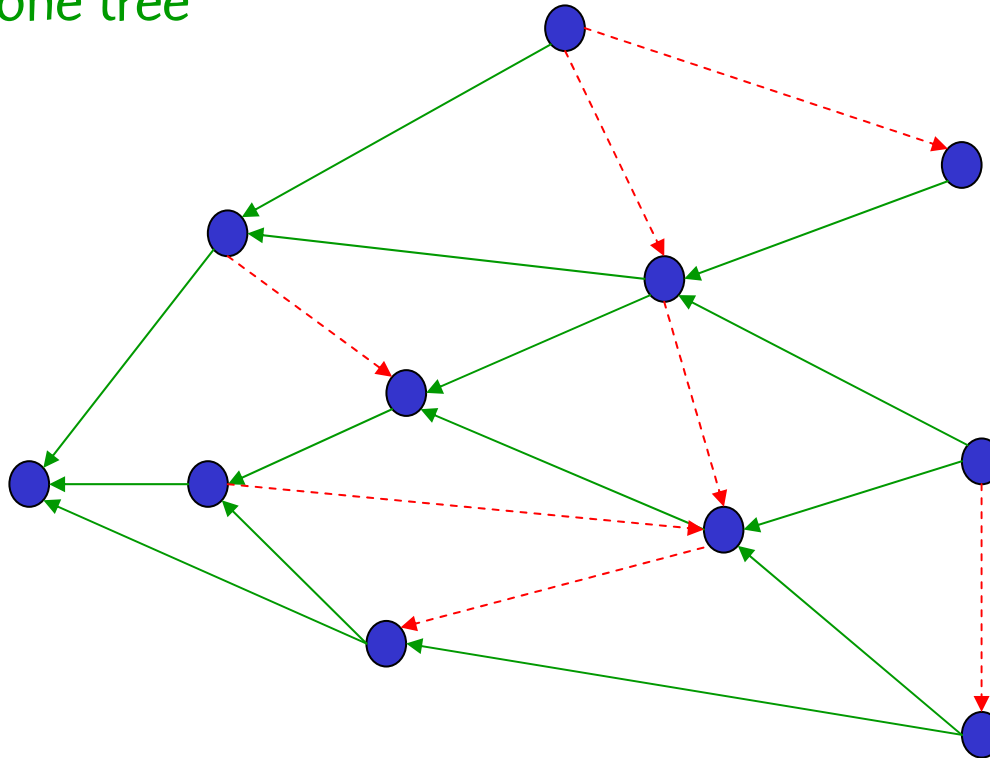
- Let  $A^*$  be the set of five arcs  $a \in A$  having largest  $\Phi_a$  values.
- Scan arcs  $a \in A^*$  from largest to smallest  $\Phi_a$ :
  - Increase arc weight, one unit at a time, in the range  $[w_a, w_a + \lceil (w_{max} - w_a)/4 \rceil]$
  - If total cost  $\Phi$  is reduced, restart local search.

# Dynamic shortest path

- In local search, when arc weight increases, shortest path trees:
    - may change completely (rarely do)
    - may remain unchanged (e.g. arc not in a tree)
    - may change partially
      - Few trees change
      - Small portion of tree changes
- } Does not make sense to recompute trees from scratch.

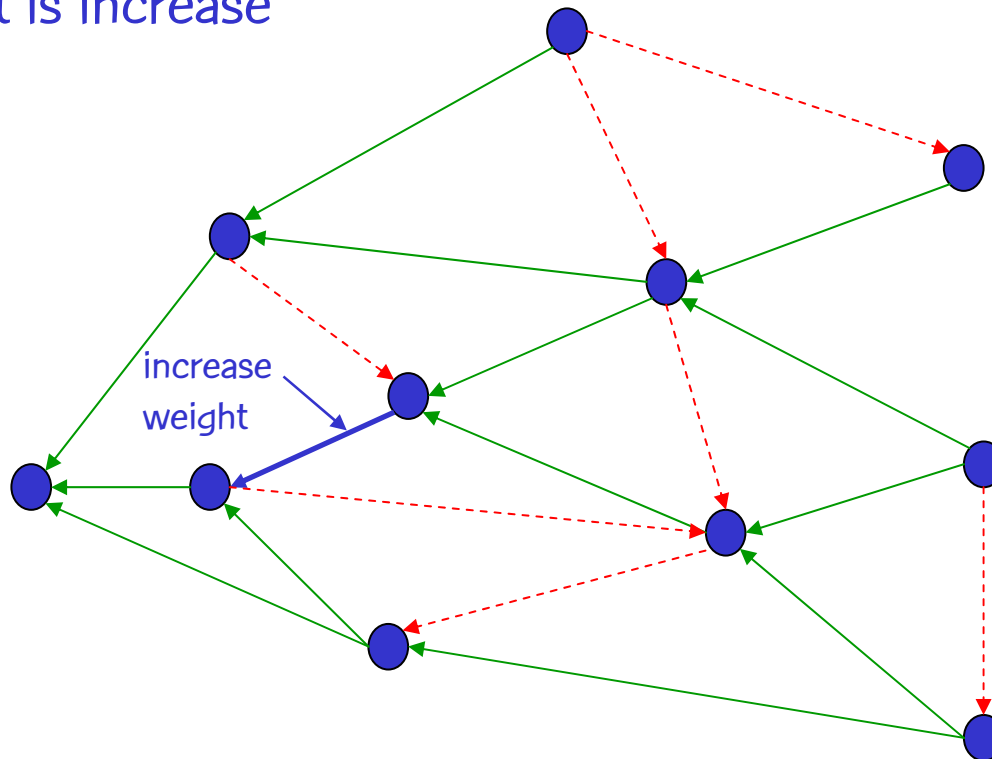
# Dynamic shortest path

Consider one tree  
at a time.



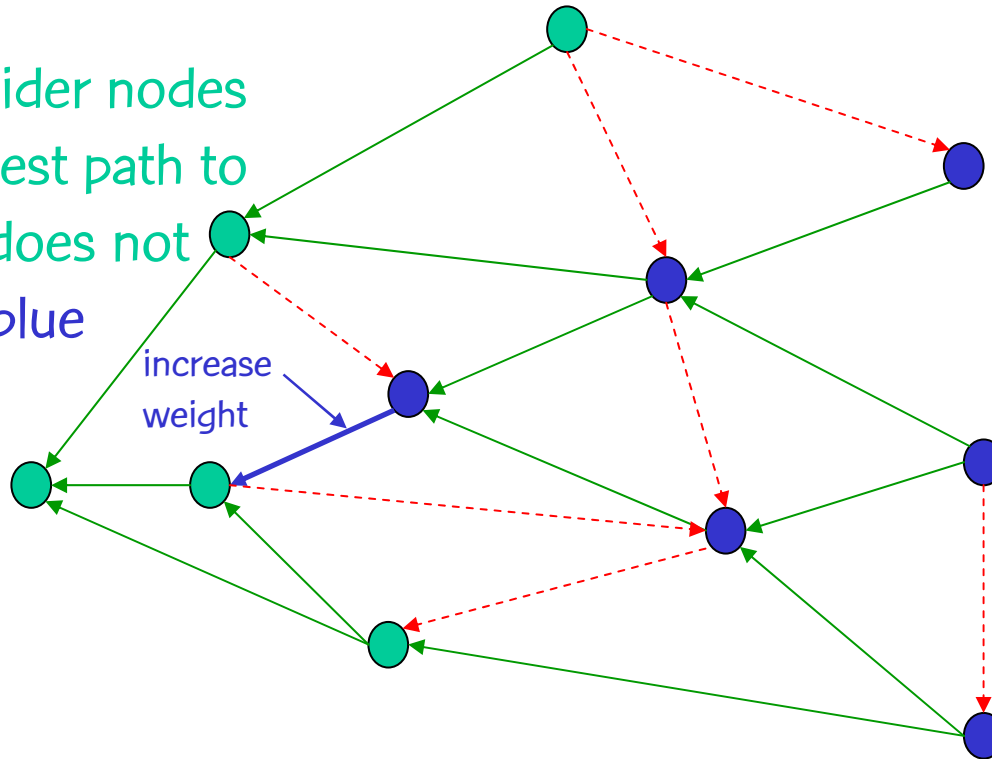
# Dynamic shortest path

Arc weight is increase  
by 1.

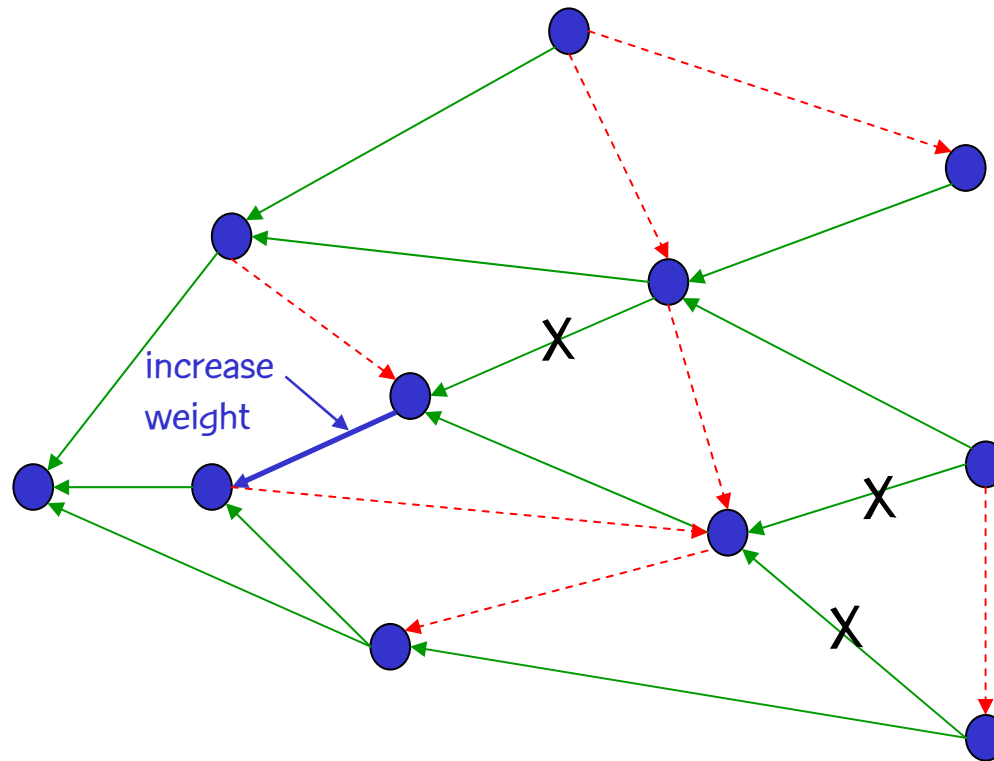


# Dynamic shortest path

Do not consider nodes  
whose shortest path to  
destination does not  
go through blue  
arc.



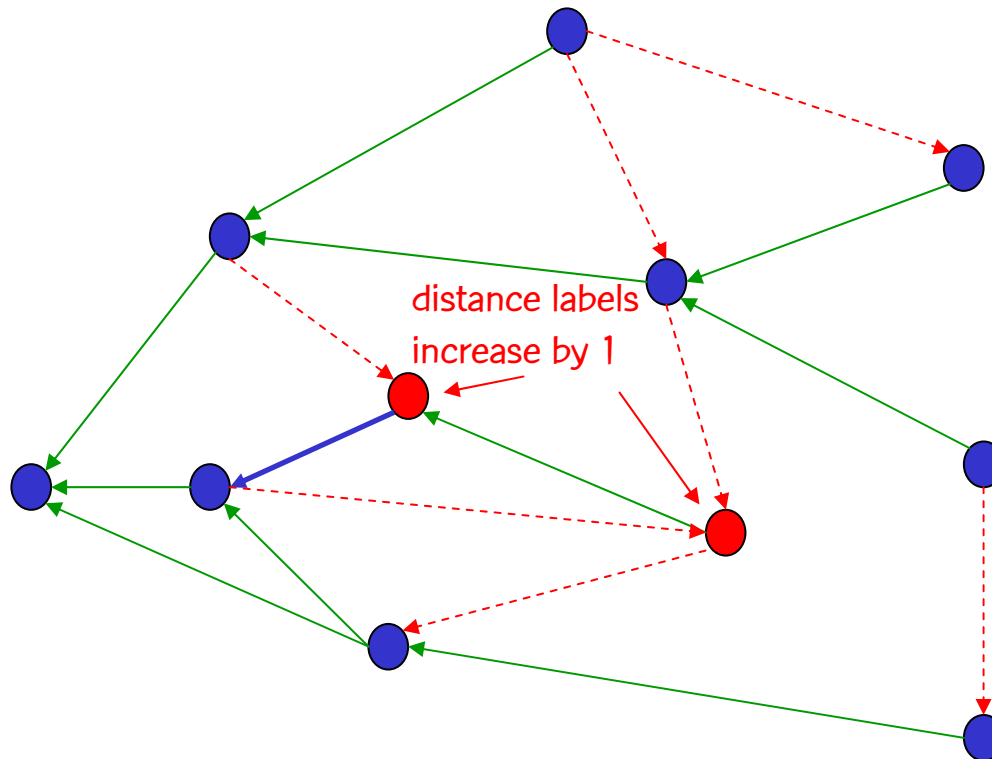
# Dynamic shortest path



Arc  $(u, v)$  is removed from tree since alternative paths from node  $u$  to the destination node exist.

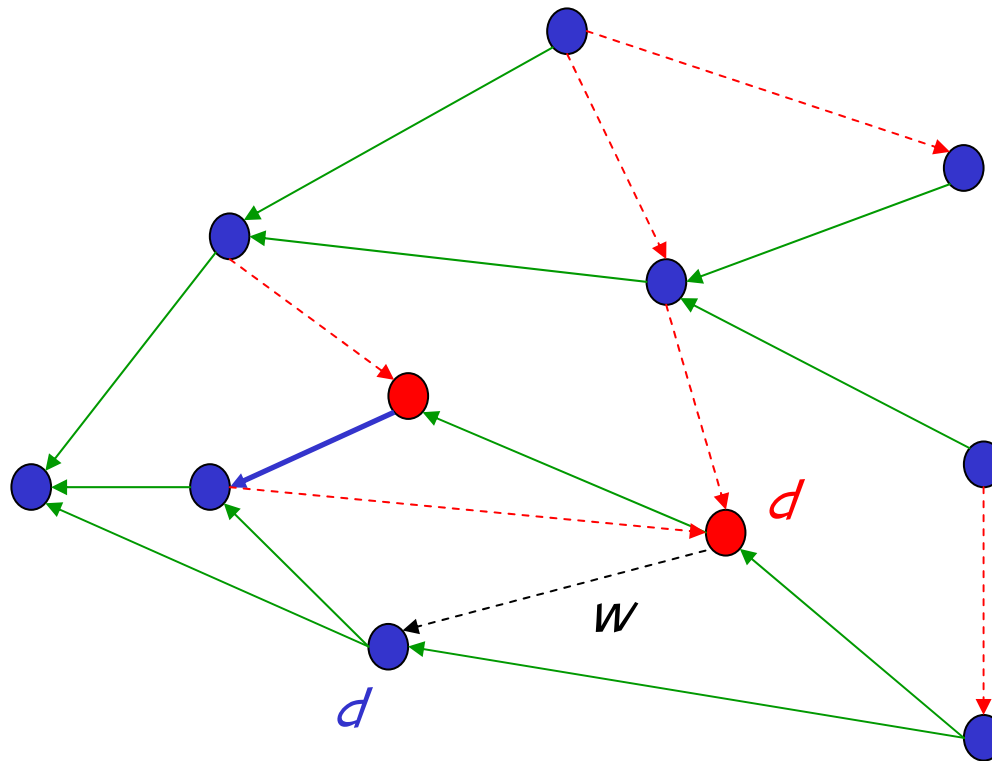


# Dynamic shortest path

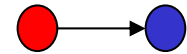


Shortest paths  
from red nodes  
must traverse  
blue arc.

# Dynamic shortest path



Test all arcs of type

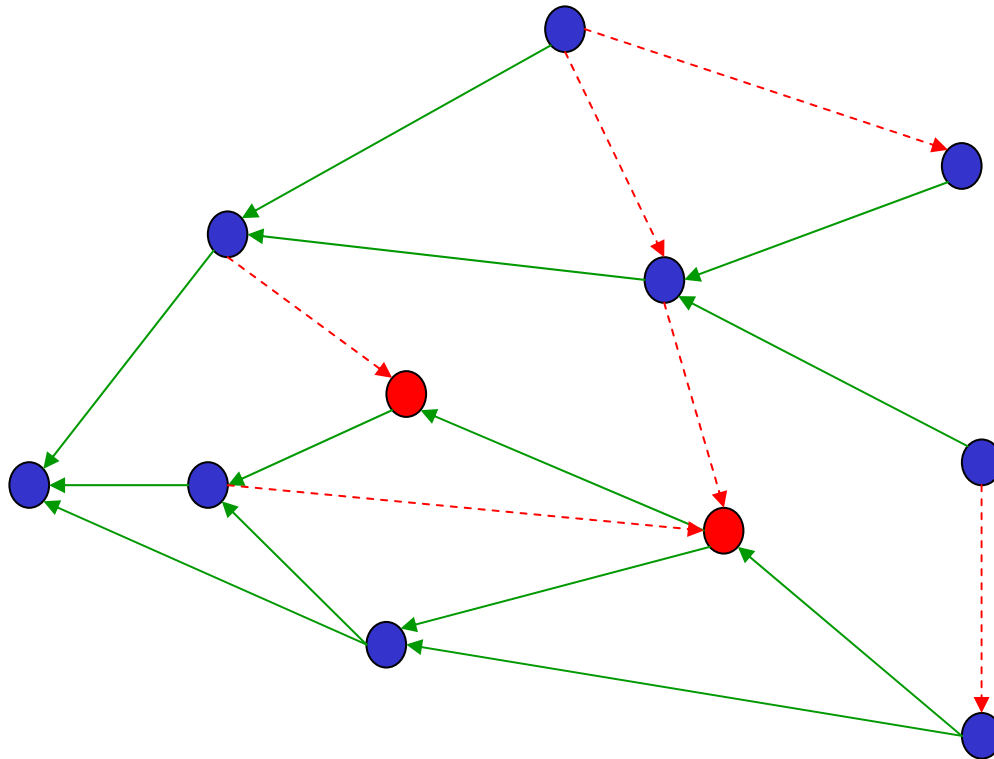


If  $d - d = w$ , then  
tree.

enters

tree.

# Dynamic shortest path

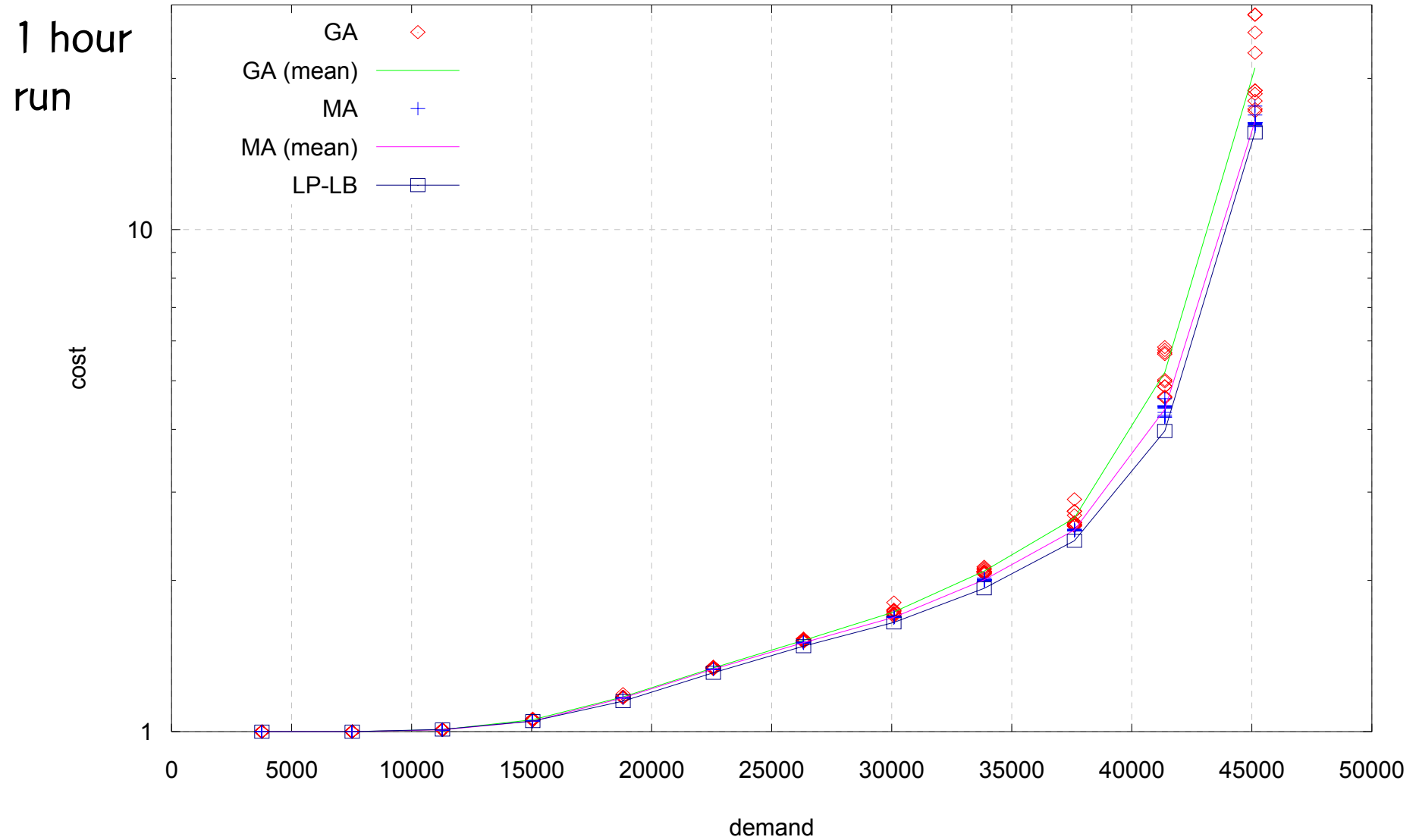


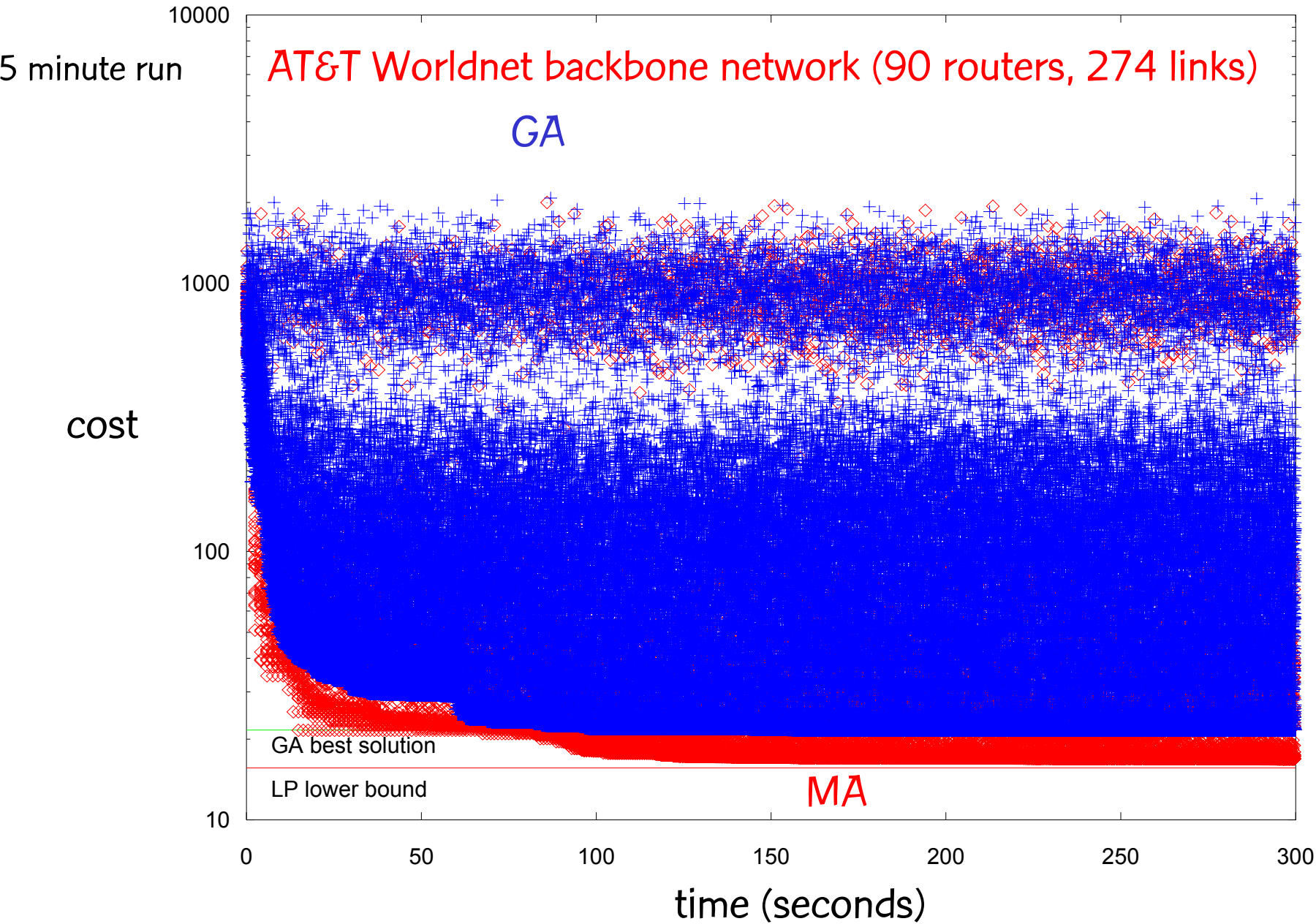
# Dynamic shortest path

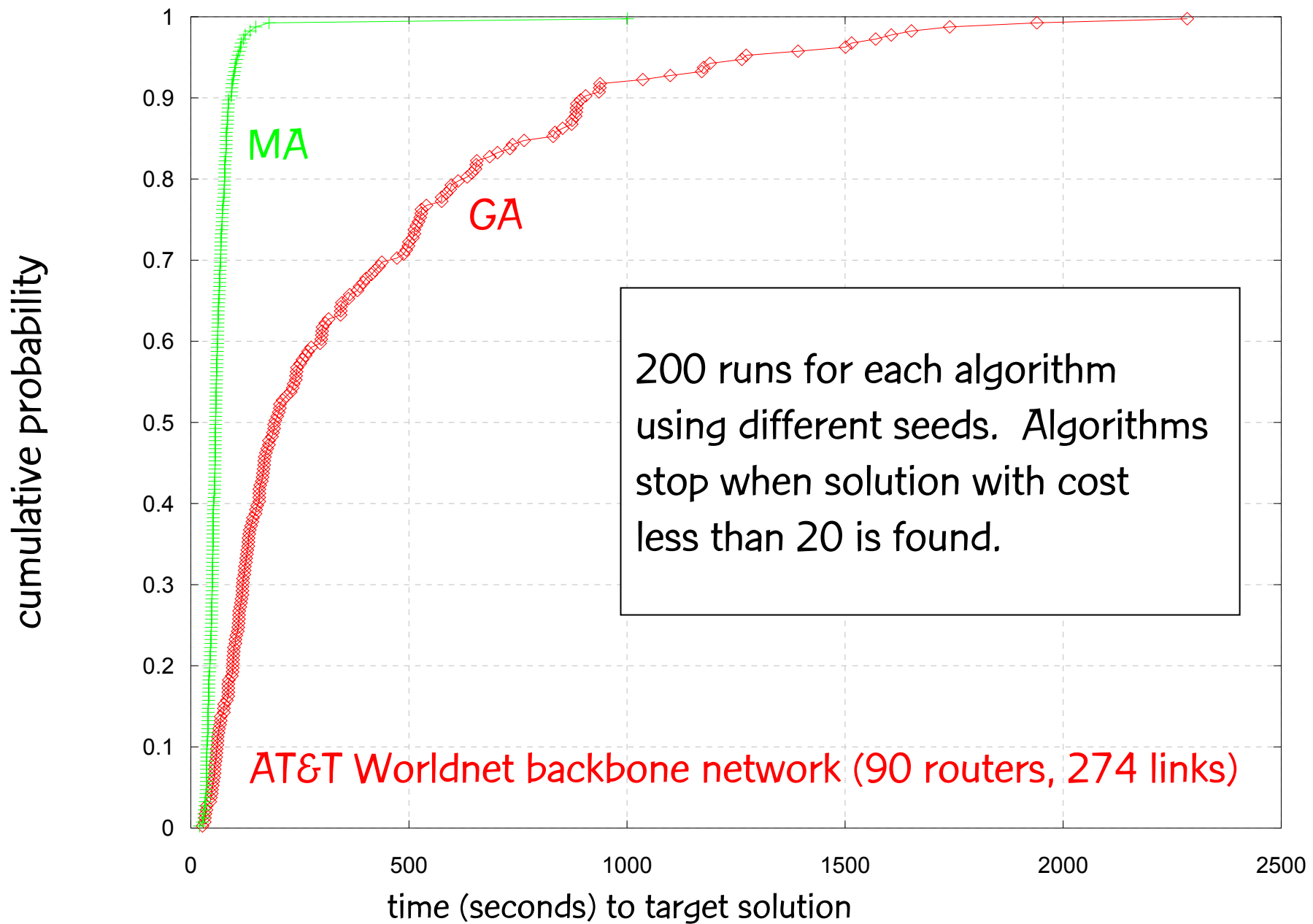
Buriol, Resende, & Thorup (2003)

- Ramalingam & Reps (1996) allow arbitrary arc weight change.
- We specialized the Ramalingam & Reps algorithm for unit arc weight change.
  - Avoid use of heaps
  - Achieve a factor of 2~5 speedup w.r.t. Ramalingam & Reps on these test problems

# AT&T Worldnet backbone network (90 routers, 274 links)

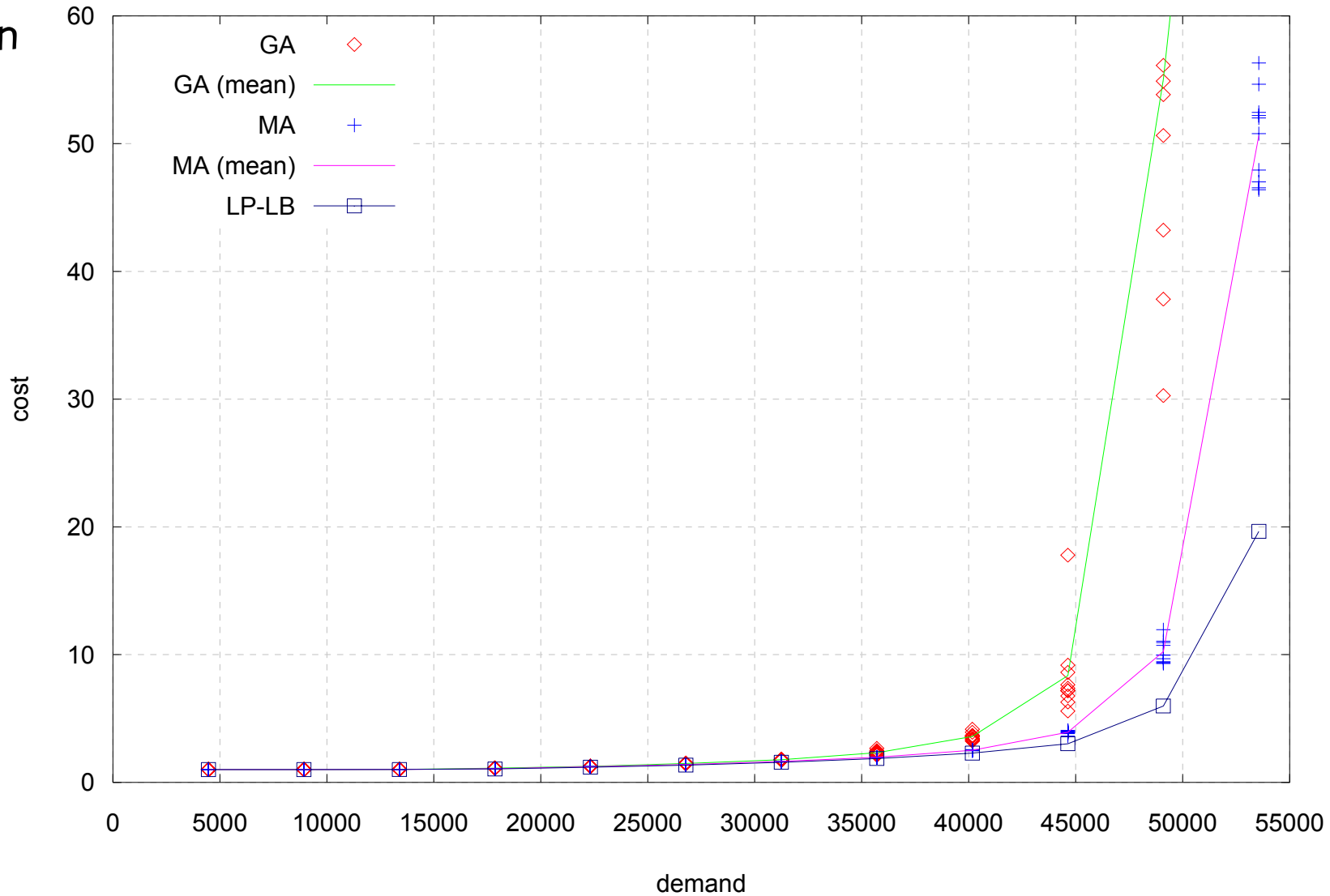






## Rand50a: random graph with 50 nodes and 245 arcs.

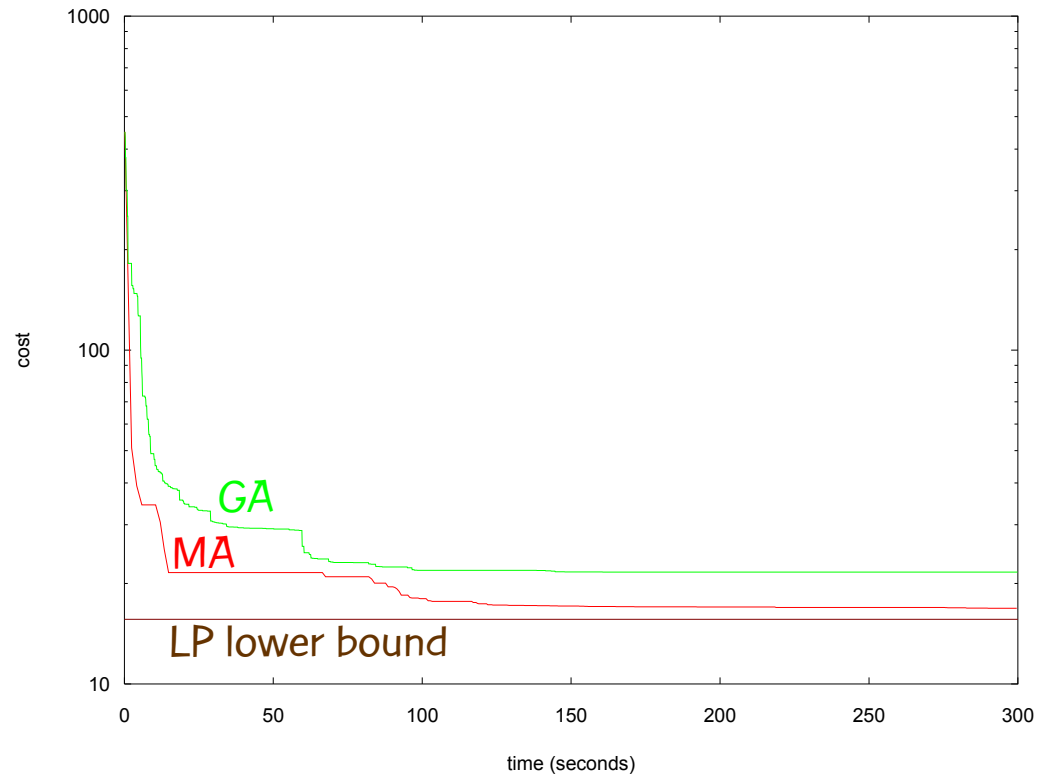
1 hour run



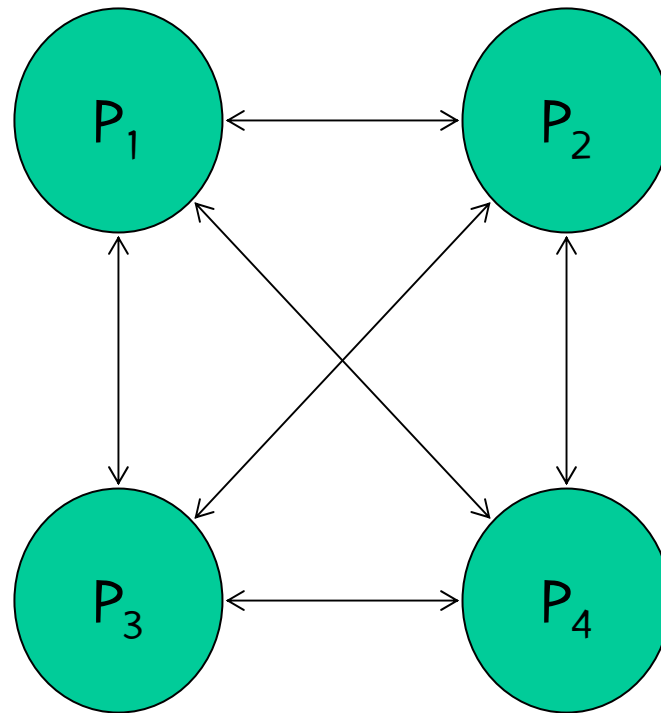


# Remark

- Memetic algorithm (MA) improves over pure genetic algorithm (GA) in two ways:
  - Finds solutions faster
  - Finds better solutions

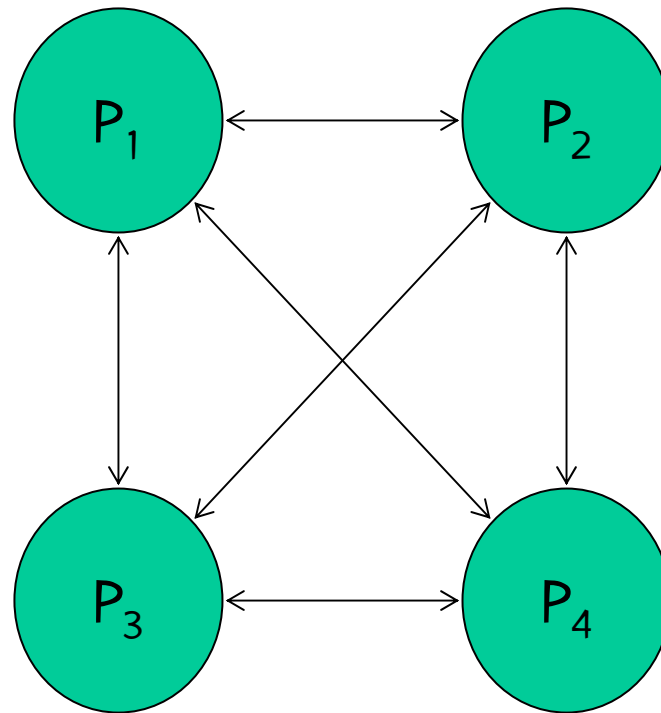


# Collaborative parallel implementation



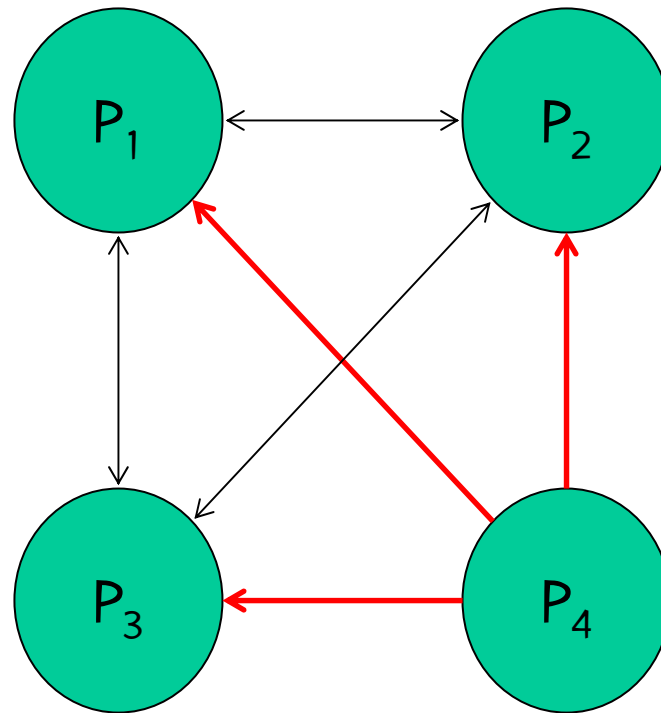
MPI: Message Passing  
Interface

# Collaborative parallel implementation



If  $P_4$  finds a new incumbent solution.

# Collaborative parallel implementation



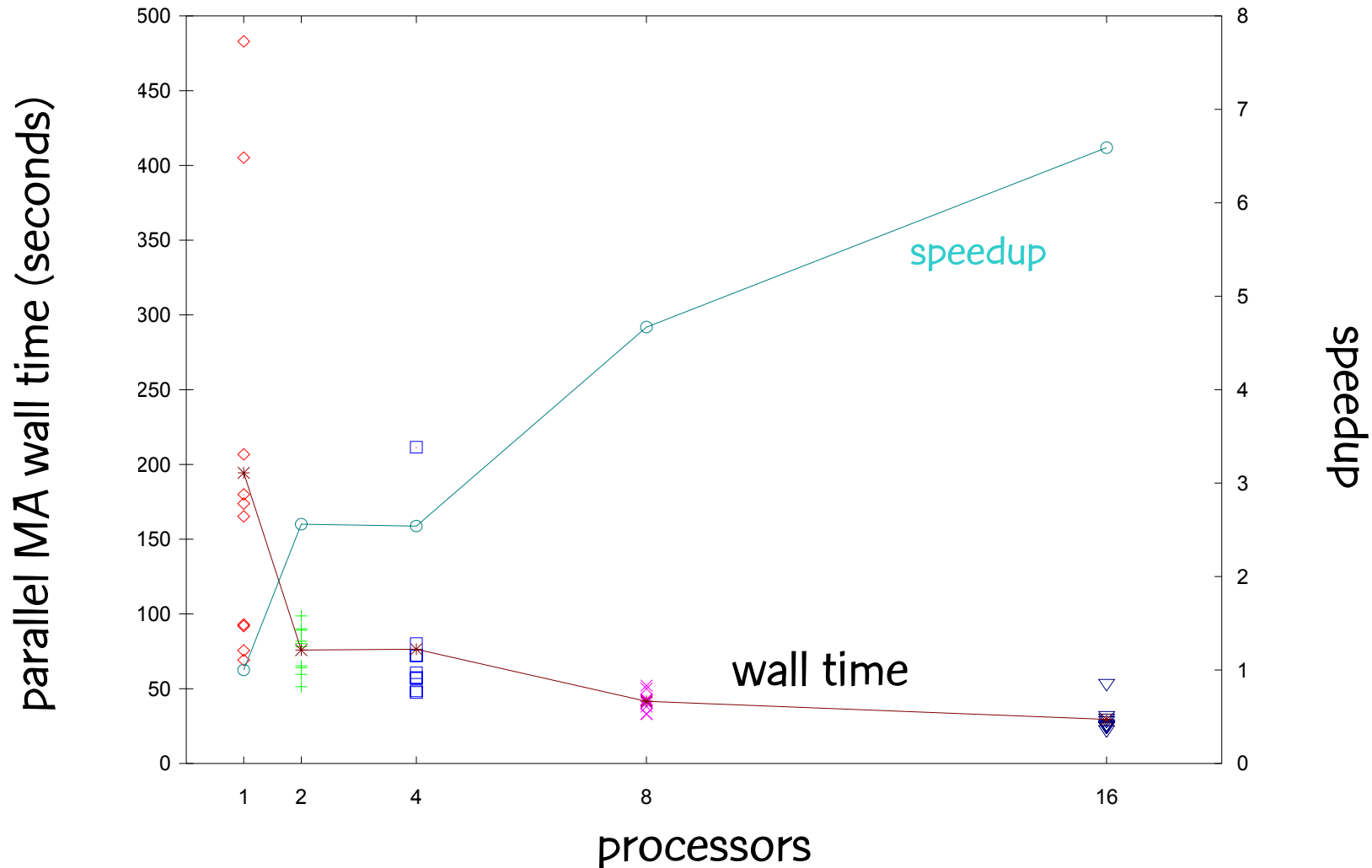
If  $P_4$  finds a new incumbent solution.  
Incumbent solution is broadcast to  $P_1$ ,  $P_2$ ,  $P_3$ .

# AT&T Worldnet backbone network (90 routers, 274 links)

demand = 45134

look4 = 18

10 parallel runs each



# Extensions

- **Network design:** Minimize total capacity  $\times$  distance of links to guarantee traffic flow subject to failures.
- **Routing:** Minimize maximum utilization subject to single link and router failures.
- **Server placement:** Locate minimum number of cache servers on network for multicast of streaming video.

# Other applications of optimization in telecommunications

- location of traffic concentrators
  - It is sometimes beneficial to concentrate traffic into a high capacity circuit and backhaul the traffic
  - Traffic is concentrated at specific nodes
  - Problem is to decide how many nodes and which
- global routing of Frame Relay service
  - To maximize the utilization of transport infrastructure one can take advantage of varying point-to-point demands due to time zone differences

# Other applications of optimization in telecommunications

- disjoint paths
  - for survivability, route several circuits between pairs of nodes on resource (node, edge) disjoint paths
  - if impossible, minimize sharing of resources
- frequency assignment
  - assign different frequencies to cellular telephone antennas to avoid interference
- SONET ring network design
  - design restorable ring networks, i.e. quickly (in less than a millisecond) react to reestablish communications



# Concluding Remarks

- we have seen a small sample of applications of optimization in telecommunications
- opportunities for optimization arise in practice all the time
- our profession call have a major impact in telecommunications

# Concluding remarks

- These slides, and papers about GRASP, path-relinking, and their telecom applications available at:  
<http://www.research.att.com/~mgcr>  
<http://graspheuristic.org>

# Handbook of Optimization in Telecommunications, P.M. Pardalos and M.G.C. Resende, Kluwer, 2004.

- Interior point methods for large-scale LP
- Decomposition methods in telecommunications
- Integer programming
- Lagrangean relaxation
- Minimum cost network flow algorithms
- Shortest path algorithms
- Multi-commodity flow in telecommunications
- Steiner tree problems in telecommunications
- Minimum spanning tree problems
- Metaheuristics
- Nonlinear programming
- Telecommunications network design
- Ring network design
- Computational large-scale linear programming
- Telecommunications access network design
- Network location in telecommunications
- Optimization issues in distribution network design
- Optimization issues in network survivability
- Virtual path design
- Network grooming
- Network reliability in telecommunications
- Optimization issues in quality of service
- Frequency assignment problem
- Optimization in cellular phone networks
- Optimization issues in web search engines
- Optimization issues in IP routing
- Network planning in telecommunications
- Pricing and equilibrium in telecommunications
- Discrete multi-commodity network flow problems and applications in telecommunications

The End