Requirements

Optimization problems, e.g. shortest path, TSP

Goals

Ant Colony Optimization as an example of biologically inspired algorithms:
Understand how the behavior of ants has inspired computer algorithms

Content

- Real ants
- Real ants: Shortest path (to the food)
- Classical path search: A*
- Ant example: Network routing
- Ant example: Travelling salesman problem
- ACO: Optimization meta heuristic
- More application examples

- Ants
  - Related to wasps and bees
  - Appeared about 100 million years ago
  - Make up 15-25% of the total bio mass
- Individual ant behaves simplistically
  - None or poor vision (i.e. light, movement)
  - Different classes of ants: workers, males, queen
  - Workers subdivided further: soldiers, builders, etc.
- Communicate through the use of pheromones
  - Chemicals produced by ants
  - Ants can lay out pheromones
  - Ants can smell pheromones laid out by others

Argentine ant,
Source: Wikipedia

❑ Deposit pheromones

- Different types for different “messages”, e.g. food, alarm, etc.
- Pheromones can be attached to the ground, food, or other objects.
- Once laid out, pheromones evaporate over time.

❑ Ants can smell

- Mixture of various pheromones
- Direction where smell comes from (two receptors, similar to human hearing)

❑ Indirect communication

- Ants can “leave a message” at some place
- Others can “pick up the message” when reaching the place
- General idea of stigmergy

Stigmergy
- A mechanism of spontaneous, indirect coordination between agents or actions, where the trace left in the environment by an action stimulates the performance of a subsequent action.
- A mechanism of indirect coordination between agents or actions, in which the aftereffects of one action guide a subsequent action.

- ❑ Each colony may have several millions of ants.
- ❑ Super-colonies of several thousand connected nests may contain hundreds of millions of ants.
- ❑ Despite the simplicity of the individual ant: complex collective behavior
  - Hunting, collecting food
  - Building nest
  - Feeding of young ants

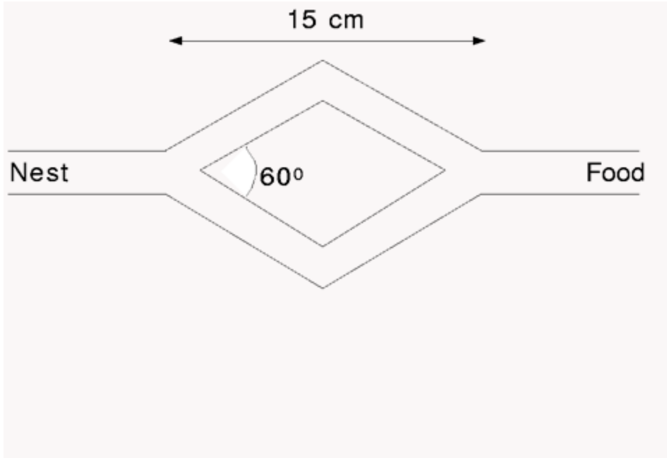
When ants search for food

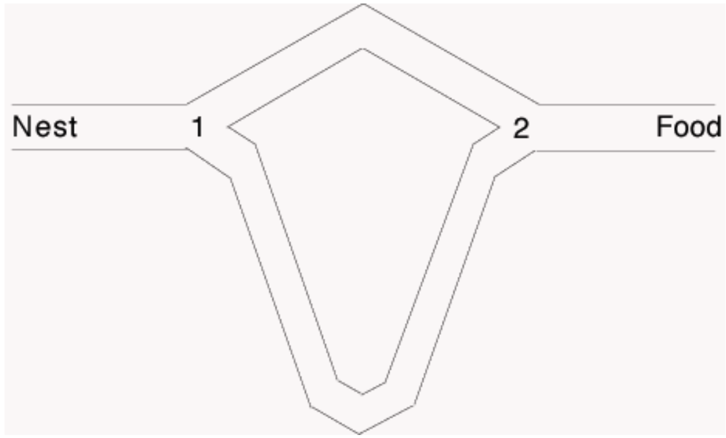
- Ants send out scouts: randomly search for food
- Return to nest with food
- Lay out pheromones on the way back
- Others follow the pheromone trail to the food source
- Pheromone trail will be strengthened.

Observation:

- There may be multiple paths leading from the nest to the food source.
- After a while ants will prefer the shortest path to almost 100% .

- Experiment conducted by Goss, 1989
- Experimental setup: Two bridges lead to the food.
  - Setup 1: Both paths have equal length.
  - Setup 2: One path is significantly longer.
- In both settings, ants eventually use a single bridge.
  - In setup 1: Bridge is chosen "randomly", if experiment is repeated several times.
  - In setup 2: Shorter bridge is eventually chosen.





- Experiment conducted by you, 2009
- Experimental setup
  - Multiple paths lead from the nest to the food
  - Paths have different lengths
- Task
  - Program ants to find the shortest path

- Example of a technical application: shortest path routing in a network

Classical Approach to Path Search

- Given a graph G = (V, E) the path search problem is defined as:
  - From a given source vertex *s …*
  - … find a path to a given destination vertex *d, …*
  - … where a path is a concatenation of edges, i.e. $(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, d)$

- Search algorithms can be divided into
  - Uninformed search
  - Informed search

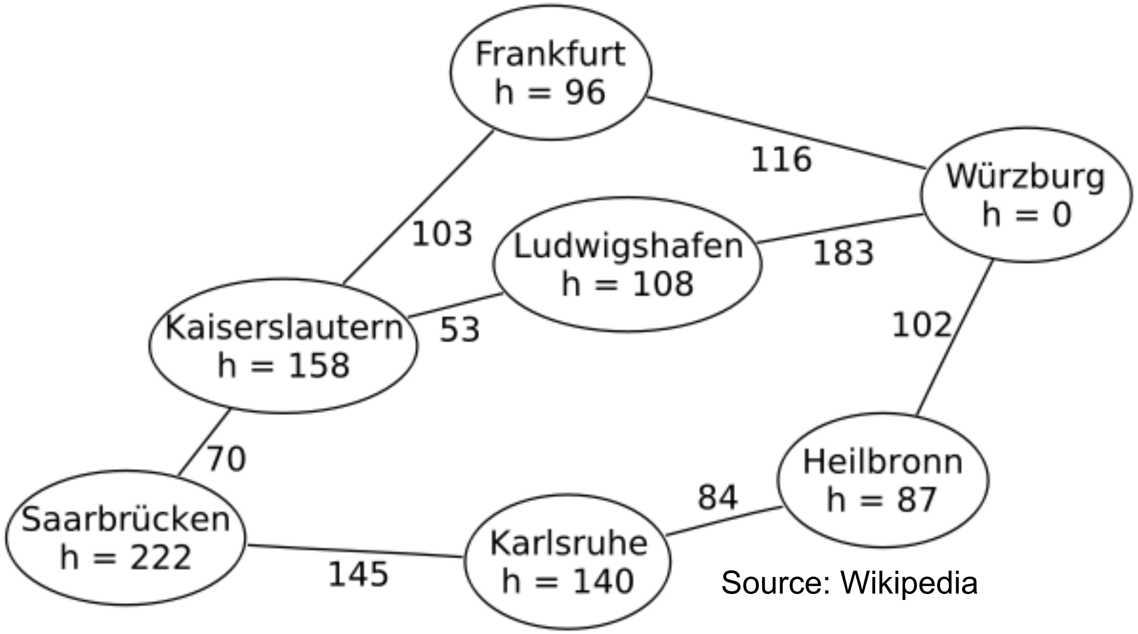
- Uninformed search: also called blind search
- Limited operations available during search:
  - List successors of a vertex
  - Distinguish between destination/goal vertex and others
- Numerous well known algorithms available, e.g.
  - Depth-first search
  - Breadth-first search
- E.g. Depth-first

```
function dfs(v):
      process(v)
      mark v as visited
      for all vertices i in successors(v)
            if !visited(i) then dfs(i)
```

- For many search problems, additional information is available:
  - Linear distance between cities
  - Routing cost of a single link
- Goal is to use this information to direct the search.
- Idea is to make an “informed guess” which vertex to expand next.
  - Best-first strategy
  - Heuristic based on available information
- Basic best-first algorithm: greedy best-first
  - Vertex-dependent heuristic for the cost of the shortest path
  - Example: Choose next city based on minimal linear distance to destination.

ISE SRA

- Find a (short) path from Saarbrücken to Würzburg
- Heuristic: Choose next vertex such that h (the linear distance to destination) is minimal.
- Algorithm will choose SB → KA → HN → WÜ (path length 331)
- But: Path SB → KL → LH → WÜ is shorter!



Source: Wikipedia

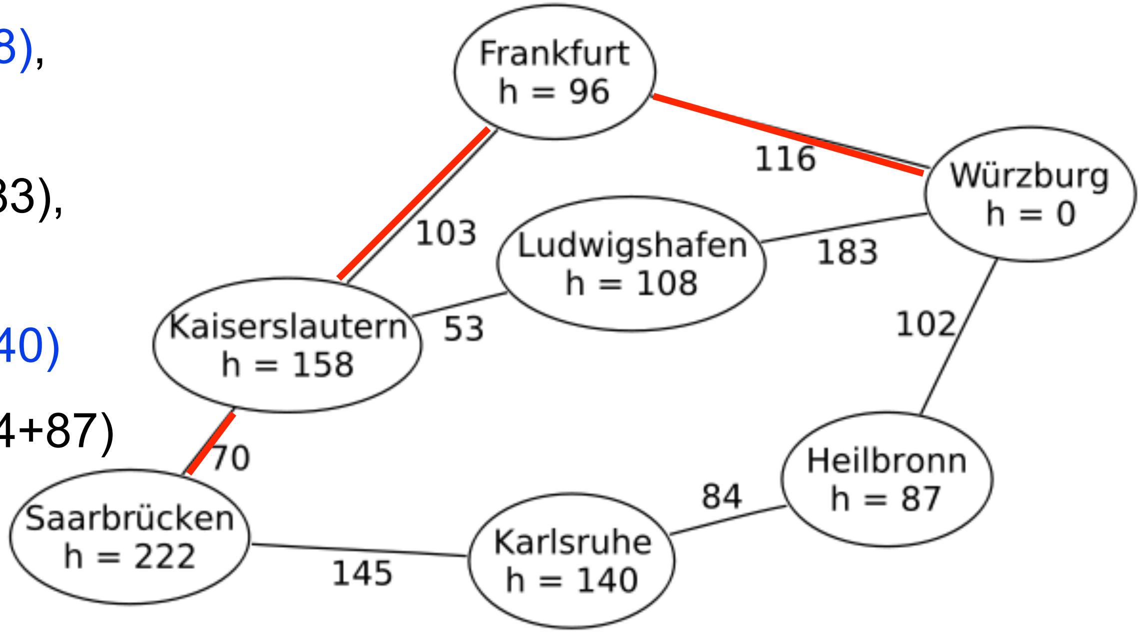
- Decision on next vertex is made based on f(x).
  - Cost to reach intermediate vertex g(x) **and**
  - Approximate cost from intermediate vertex to destination h(x)
  - f(x) = g(x) + h(x)
- Assumption: h(x) is monotonic or consistent.
  - h(x) never overestimates the real cost.
  - For every vertex k and successor k' of k, the following holds:
    h(k) ≤ c(k, k') + h(k')

    i.e. estimated cost from k to destination, h(k), is less or equal to the actual cost from k to k', c(k,k'), plus the estimated cost from k' to the destination, h(k')
- It can be shown that A* is optimal, if h(n) is monotonic.

A* 1968: Peter Hart, Nils J. Nilsson und Bertram Raphael

- ❑ Find a (short) path from Saarbrücken to Würzburg
  - -> Vertices are evaluated according to f(x) = g(x) + h(x)
- ❑ Algorithm will choose SB → KL → F → WÜ (path length 289)

1. (SB, 0)
2. (KL, 70+158), (KA, 145+140)
3. (F, 70+103+96), (LH, 70+53+108), (KA, 145+140)
4. (F, 70+103+96), (WÜ, 70+53+183), (KA, 145+140)
5. (WÜ, 70+103+116), (KA, 145+140)
6. **(WÜ, 70+103+116)**, (HN, 145+84+87)



http://de.wikipedia.org/wiki/A*-Algorithmus
http://www-m9.ma.tum.de/material/de/spp-a-star/

A* algorithm (1)

declare openlist **as** PriorityQueue **with** Nodes // *Prioritätenwarteschlange*
declare closedlist **as** Set **with** Nodes

A* algorithm (2)

```
program a-star
   // Initialisierung der Open List, die Closed List ist noch leer (die Priorität bzw. der f Wert des
  // Startknotens ist unerheblich)
   openlist.enqueue(startknoten, 0)
   // diese Schleife wird durchlaufen bis entweder
   // - die optimale Lösung gefunden wurde oder
   // - feststeht, dass keine Lösung existiert
   repeat
      // Knoten mit dem geringsten f Wert aus der Open List entfernen
      currentNode := openlist.removeMin()
      // Wurde das Ziel gefunden?
      if currentNode == zielknoten then
         return PathFound
      // Der aktuelle Knoten soll durch nachfolgende Funktionen nicht weiter untersucht werden
     //  damit keine Zyklen entstehen
      closedlist.add(currentNode)
      // Wenn das Ziel noch nicht gefunden wurde: Nachfolgeknoten des aktuellen Knotens auf die
     // Open List setzen
      expandNode(currentNode)
   until openlist.isEmpty()
   // die Open List ist leer, es existiert kein Pfad zum Ziel
   return NoPathFound
end
```

A* algorithm (3)

```
// überprüft alle Nachfolgeknoten und fügt sie der Open List hinzu, wenn entweder
// - der Nachfolgeknoten zum ersten Mal gefunden wird oder
// - ein besserer Weg zu diesem Knoten gefunden wird
function expandNode(currentNode)
    foreach successor of currentNode
        // wenn der Nachfolgeknoten bereits auf der Closed List ist - tue nichts
        if closedlist.contains(successor) then
            continue
        // g Wert für den neuen Weg berechnen: g Wert des Vorgängers plus
        // die Kosten der gerade benutzten Kante
        tentative_g = g(currentNode) + c(currentNode, successor)
        // wenn der Nachfolgeknoten bereits auf der Open List ist,
        // aber der neue Weg nicht besser ist als der alte - tue nichts
        if openlist.contains(successor) and tentative_g >= g(successor) then
            continue
        // Vorgängerzeiger setzen und g Wert merken
        successor.predecessor := currentNode
        g(successor) = tentative_g
        // f Wert des Knotens in der Open List aktualisieren bzw. Knoten mit f Wert in die Open List einfügen
        f := tentative_g + h(successor)
        if openlist.contains(successor) then
            openlist.decreaseKey(successor, f)
        else
            openlist.enqueue(successor, f)
    end
end
```

```
program a-star( source, destination)
   // priority queues for known vertices (open) and finally
   //evaluated vertices
   var openqueue, closedqueue;
   openlist.enqueue(source, 0)  // second parameter: priority f
   repeat
       // select known vertex with lowest estimate f next
       currentNode := openlist.removeMin()
       if (currentNode == destination) then return PathFound
       // examine successors (shown next)
       expandNode(currentNode)
       // currentNode has been fully examined
       closedlist.add(currentNode)
   until openlist.isEmpty()
   // openqueue is empty: no path to destination
return NoPathFound end
```

```
// examine all successors of vertex and add them to the openlist, iff
// - successor found for the first time or shorter way has been found
function expandNode(currentNode)
      foreach successor of currentNode
      if closedlist.contains(successor) then continue
      // calculate f for the new path
      f := g(currentNode) + c(currentNode, successor) + h(successor)
      // if successor is on openlist and new estimate is larger: skip
      if openlist.contains(successor) and f > openlist.find(successor).f
            then continue
      // update f-estimate of successor in openlist or insert it
      if openlist.contains(successor) then
      openlist.decreaseKey(successor, f)
      else openlist.enqueue(successor, f)
      end
end
```

- Artificial ants are simple agents.
- Pheromones: Values attached to a location
- Location may be modeled depending on the problem, e.g.
  - (x,y)-coordinate
  - Edges of a graph
- Each agent has a local memory, e.g. the path travelled so far.
- Each agent has a probabilistic behavior: depends on the perception of pheromones
- Typically, agents will follow those paths with a higher probability that have a higher pheromone density.

- ACO: Metaheuristic presented by Marco Dorigo
- Ants build incremental solutions concurrently and asynchronously.
- Stochastic local decision policy uses heuristic information and pheromone trails.
- Ants locally evaluate their (partial) solution and deposit pheromones.
- EvaporatePheromone: models the fading of pheromones over time.
  - Avoids a too rapid convergence
  - A form of forgetting in favor of exploration
- DaemonActivities: Actions that cannot be performed by a single ant
  - Drop extra pheromone for best solution
  - Called offline pheromone update

```
procedure ACO
  metaheuristic
    ScheduleActivities
        ManageAntsActivity
        EvaporatePheromone
        DaemonActions
    end ScheduleActivities
end ACO metaheuristic
```

- Well this seems all very ♋♌⬧⧫❒♋♍⧫! What can I do with it???

- Problem:
  Find shortest (multi-hop) route between pairs of nodes in a packet-switched network.

- “Classic” approach, e.g. distance vector routing
  - Routing tables: rows for each destination; columns for each neighbor
  - tab(*d*, *n*) contains a cost measure (e.g. number of hops) to route to node *d* via neighbor *n.*
  - Newly acquired routing information is broadcasted to neighbors.
  - Routing tables are adapted according to incoming messages.

- Network Routing using Ants, Di Caro and Dorigo, 1998
- Artificial ants (mobile agents)
  - May travel through the network
  - Goal: Find and mark good routes
- Links in the network can be marked with artificial pheromones.
- Route quality is measured by the length of the route (travel time) and the stability of the route (variance of time).
- Data packets are routed probabilistically according to pheromone values.

- Data structures on each node contain:
  - Probabilistic routing table
  - Traffic statistics (local information)
- Routing Table
  - One row for each neighbor
  - One column for each node in the network (destination)
  - tab(n,d) contains the probability for routing a packet to destination d through neighbor n
- Traffic statistics
  - Stores statistics about trip durations (e.g. mean and variance)
  - Used when routing probabilities are modified

- Traffic consists of three classes of packets
  - Forward ants: ants looking for a route to a destination
  - Backward ants: ants returning from (successfully) searching for a route
  - Data packets (payload)
- Route selection
  - Probabilistic, according to routing table
- Update of the probabilities in the routing table
  - Increase the probability for links that performed well

- ❑ Task of forward ants (FA): Find a (shortest) path from a source to some destination.
- ❑ FAs are either sent:
  - … periodically from each node to all other nodes (proactively)
  - … or only to destinations requested by the application (reactively)
- ❑ FAs collect travel time
  - Sent through the same queues as data packets
  - Experience same delays as data packets
- ❑ FAs collect travelled route.
  - Hop-list in local memory
  - Used to detect routing cycles (→ poor ant dies on long cycles)

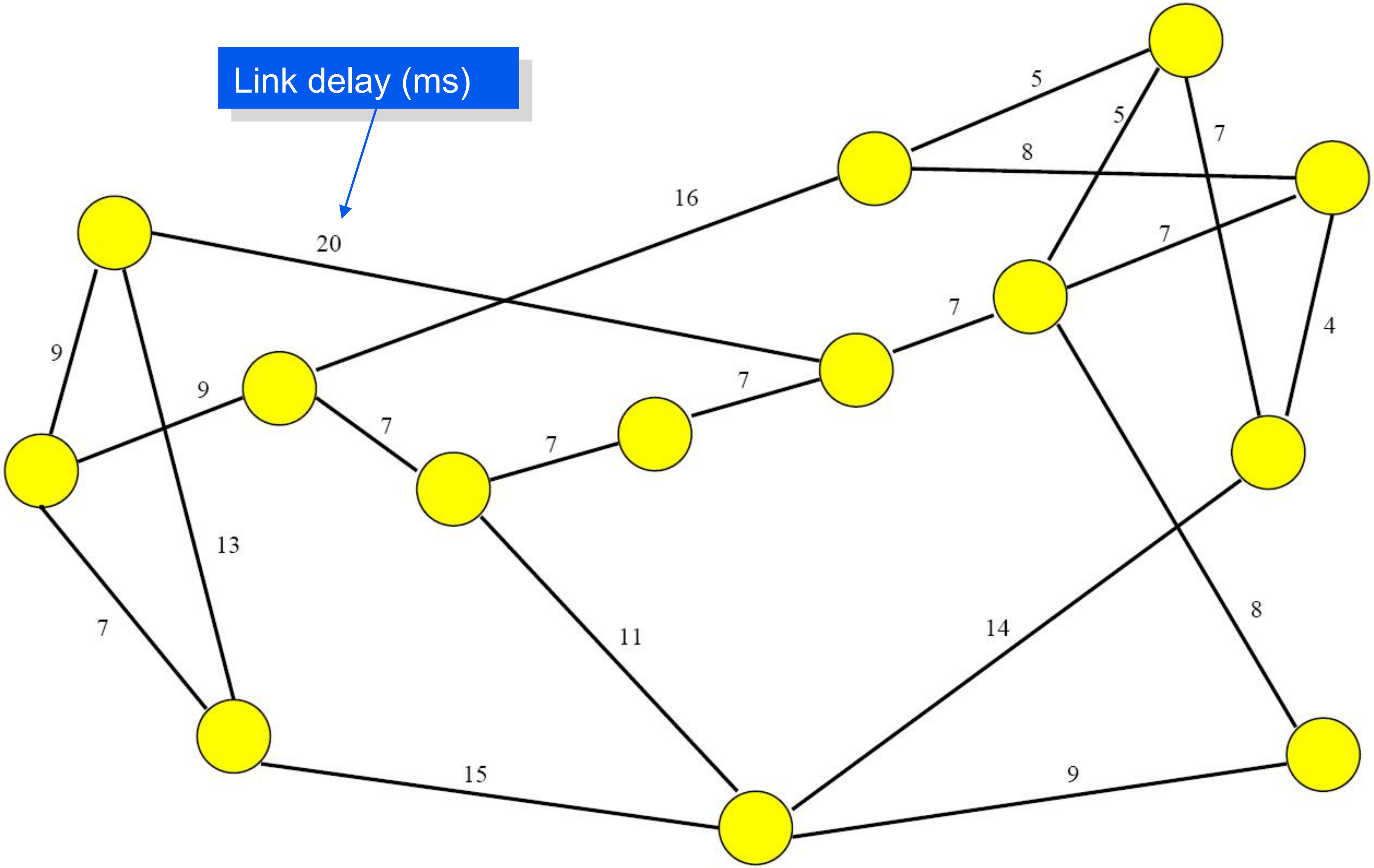
- Probabilistic hop-decision is done
  - according to routing table and
  - according to link queue lengths.
- High chance of taking a (previously) good next hop
- With a small exploration probability next hop is chosen randomly with uniform distribution: Chance of adapting to changes!
- If ant has already traveled through the chosen node: apply exploration!

- When a FA reaches its destination: turns into a backward ant (BA)
- BA takes reverse path back to the source (with high priority)
- At each intermediate node i_k on the path (i_0, …, i_n) the routing probabilities are updated.

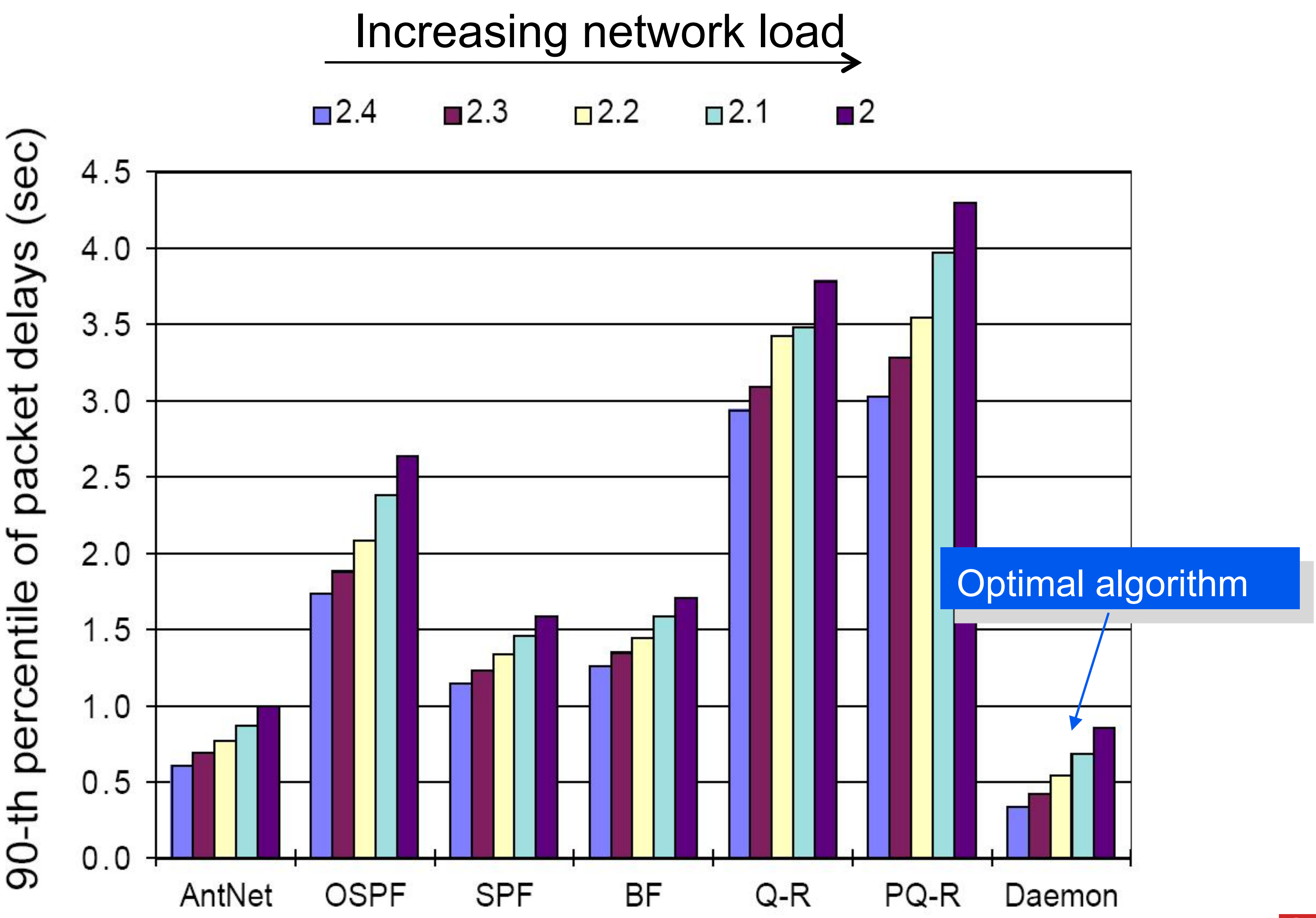
  -> Where i_0 is the source node and i_n is the destination node
- The routing probabilities for i_k on i_{k-1} is increased as a function of travel time (lower time → higher increase)
- Probability of all other neighbors of i_{k-1} is decreased.
- Leads to a higher probability of choosing that path again next time, if the travel time is short.

ISE SRA

AntNet: Performance



AntNet: Performance (2)



Source: Di Caro, 2004

- ❑ Shortest path search is an interesting polynomial problem.
- ❑ Lets see how the ants do in a more difficult situation. ☺
- ❑ The Travelling Salesman Problem (TSP)
  - Given a set of cities and the cost for travelling between each pair of cities
  - Problem: What is the least-cost round-trip
    - that visits each city once and
    - returns to the starting city?
  - Solution space: ½ (n-1)!, n>2, (number of Hamiltonian cycles in a complete graph)
  - NP-hard
- ❑ TSP is a classical benchmark (not only) for ant-based heuristics.

Optimaler Reiseweg eines Handlungsreisenden durch die 15 größten Städte Deutschlands. Die angegebene Route ist die kürzeste von 43.589.145.600 möglichen.

- Ant System: proposed by Dorigo et al., 1991
- Ant-based heuristic to approximate TSP
- Initially a random vertex (city) on the graph is chosen for each ant.
- Each ant remembers the route travelled so far (start city when ant “is born”).

- When ant *k* is at city *i*, it chooses to go to a yet unvisited city *j* with

Pheromone information

A priori heuristic information

$$\eta_{ij} = {}^{1}/_{d_{ij}}$$

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}$$

α, β: determine the relative influence of pheromone and heuristic information.

Set of neighbours not yet visited

- If α is 0: greedy search based on heuristic information
- If β is 0: only pheromones are used
  - Typically results in early stagnation
  - Sub-optimal results

- After each ant has finished a tour, update pheromone values for all edges (i,j):

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

Evaporation rate of pheromone, 0 < ρ ≤ 1

Amount of pheromone deposited by an ant k on edge (i,j)

- And the amount of pheromone deposited by an ant k on edge (i,j) is:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{if } (i,j) \text{ is used by ant } k \\ 0 & \text{otherwise} \end{cases}$$

Length of the tour found by ant k

- The original Ant System performs well for rather small problem sizes (up to approx. 40 vertices/cities)
  - Performance in terms of “solution quality”
  - Computational complexity per iteration is in $O(mn^2)$, where m is the number of tours/ants and n the number of cities!
- Numerous improvements to the Ant System have been proposed, e.g.
  - Elitist strategy
  - Max-Min-Ant System
  - Ant Colony System (ACS)

- Elitist strategy introduces a global action after ants have finished an iteration (daemon action in the metaheuristic).
- Idea: Give an extra amount of pheromones to the best tour found so far (over all iterations).
- Depending on the length of the best tour T^{gb}, pheromones of a number *e* of elitist ants is spread over all edges of the best tour.

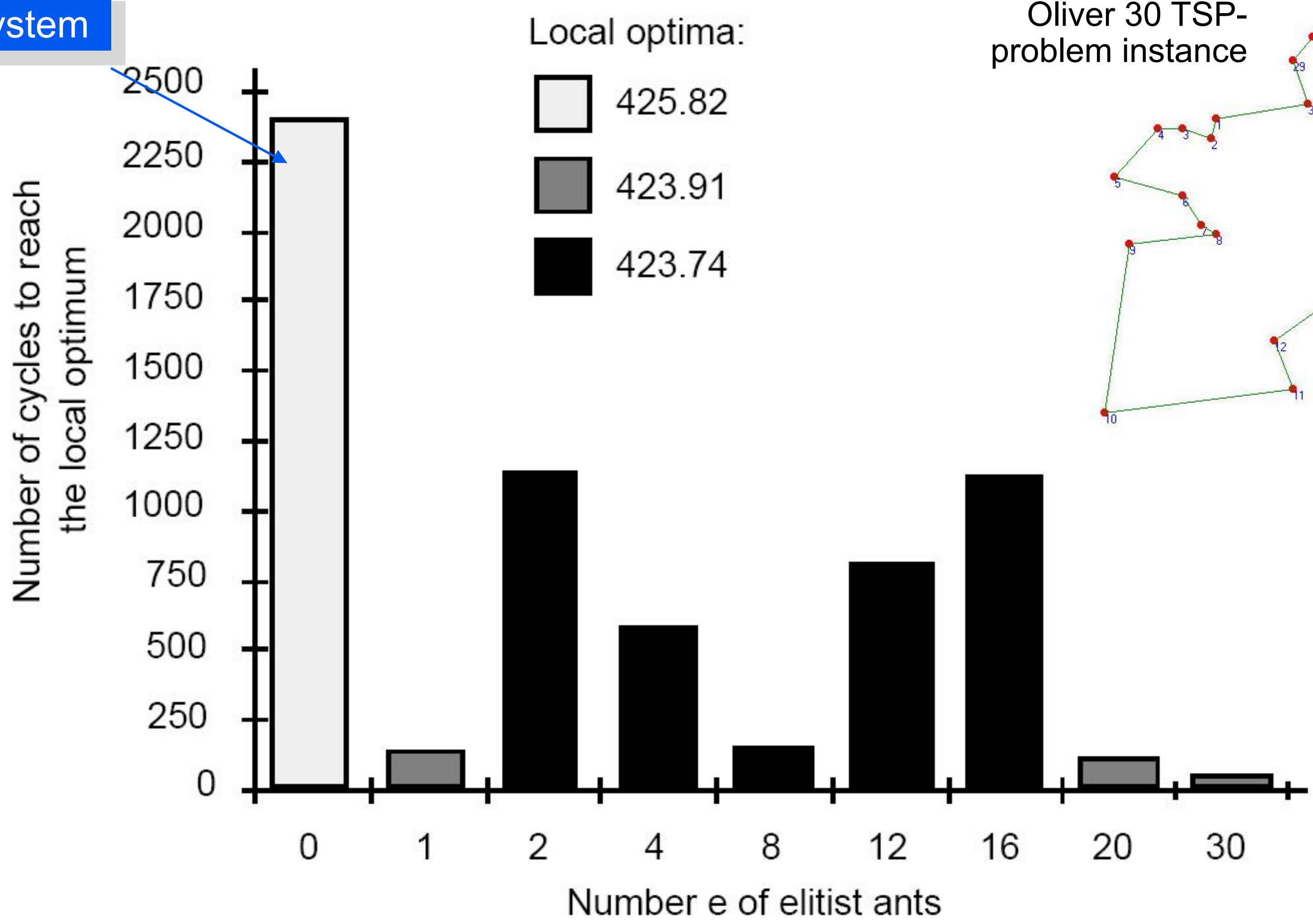
$$\Delta\tau_{ij}^{gb}(t) = \begin{cases} e / L^{gb}(t) & \text{if } (i,j) \in T^{gb} \\ 0 & \text{otherwise} \end{cases}$$

- Similar idea used in the AS_{rank} extension: Only the best *w* tours per iteration deposit pheromones (weighted by rank).

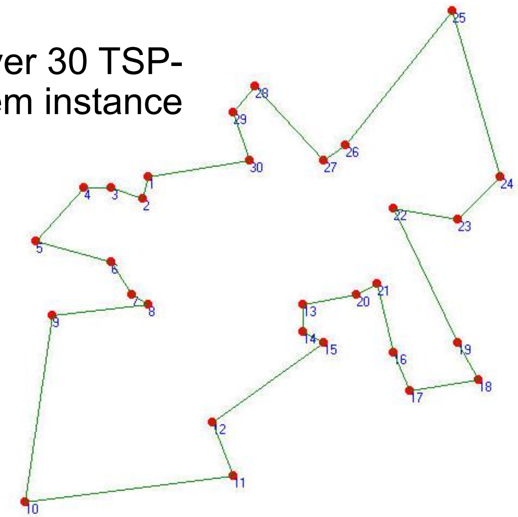
Speeds up convergence!

Elitist Strategy: Performance

Initial ant system



Oliver 30 TSP-problem instance



- Max-Min Ant System (MMAS), Stützle & Hoos, 2000
- Uses elitist strategy (either global or iteration best)
- Value of the pheromone trails limited to $[\tau_{min}, \tau_{max}]$
- Trails are initialized to the upper limit τ_{max} . → Enables higher exploration at algorithm start.
- Pheromone trails are reinitialized each time no improvement is found for a certain number of iterations.

- See also for comparison: Simulated annealing!

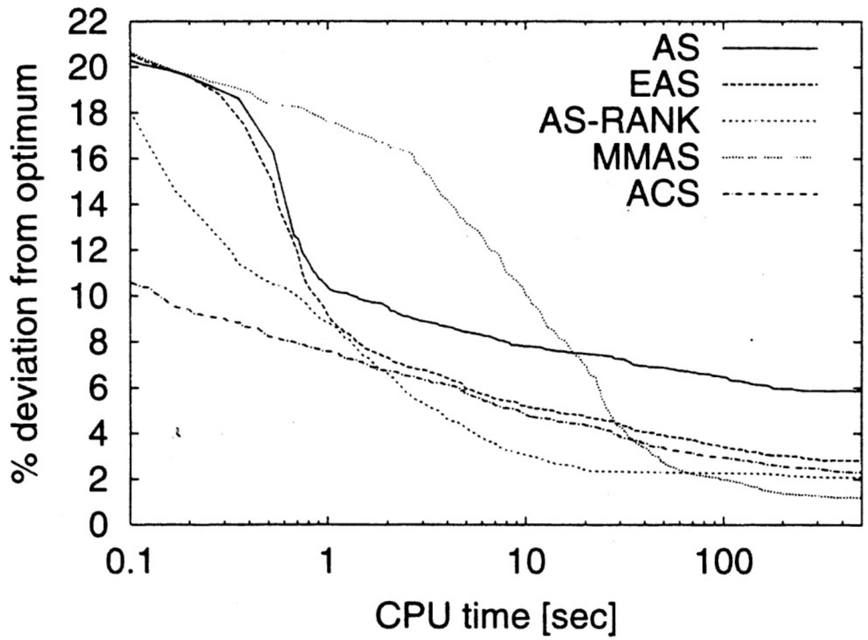
- ACS also strongly inspired by AS, three major changes
- Stronger exploitation of search experience
- Pheromone evaporation and deposit only on edges belonging to the best-so-far tour

Best-so-far tour

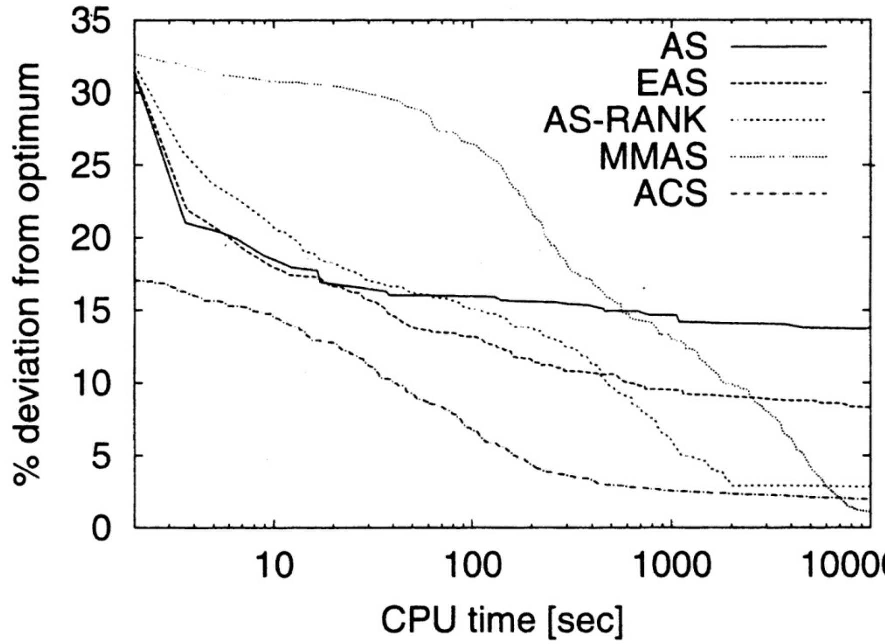
  - $$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \rho\Delta\tau_{ij}^{bs} \qquad \forall (i,j) \in T^{bs}$$
  - Reduces computational complexity of pheromone update from $O(n^2)$ to $O(n)$
- Each time an ant uses an edge, it removes some pheromones.

Performance Comparison: Runtime vs. quality

TSPLIB: d198



TSPLIB: rat783



Source: Dorigo & Stützle, 2004

Further Reading

- Marco Dorigo and Thomas Stützle: Ant Colony Optimization, MIT Press, 2004
- Gianni Di Caro, Marco Dorigo: AntNet: A Mobile Agents Approach to Adaptive Routing, Technical Report IRIDIA 97-12