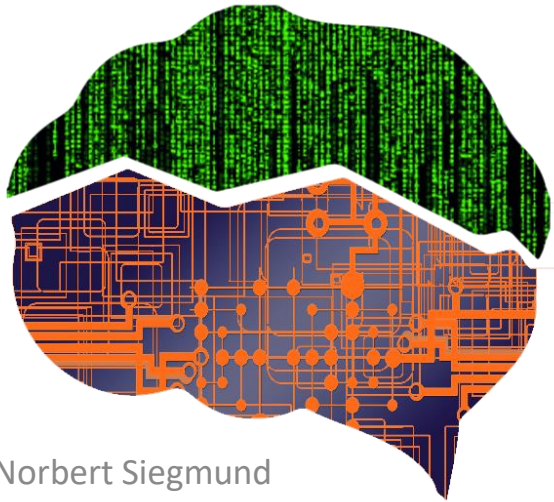


# Search-Based Software Engineering

Combinatorial Optimization



Prof. Dr.-Ing. Norbert Siegmund  
Intelligent Software Systems

Bauhaus-Universität  
Weimar

# Recap I

---

- Multi-objective optimization: How to define objective?
  - Weighted sum
  - Preference ranking
  - Entrance-based tournament selection
- Pareto front
  - Pareto dominance
  - Front rank
- Non-Dominated sorting

# Recap II

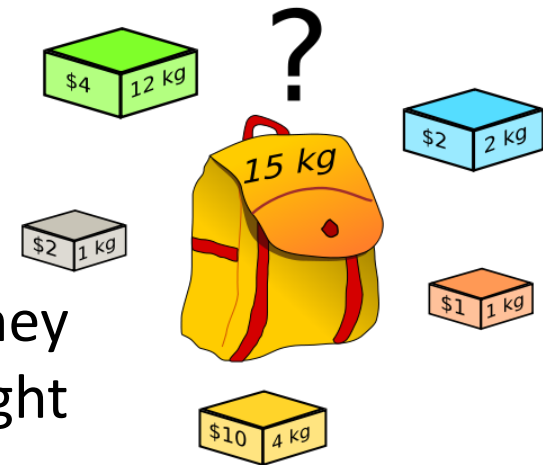
---

- Sparsity
- NSGA-II
  - Non-dominated sorting genetic algorithm II
  - Keep all the best known individuals
  - Use sparsity to spread out in the optimization
- Pareto strength and wimpiness
- SPEA2

# Combinatorial Optimization

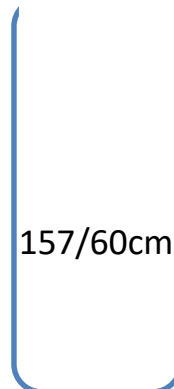
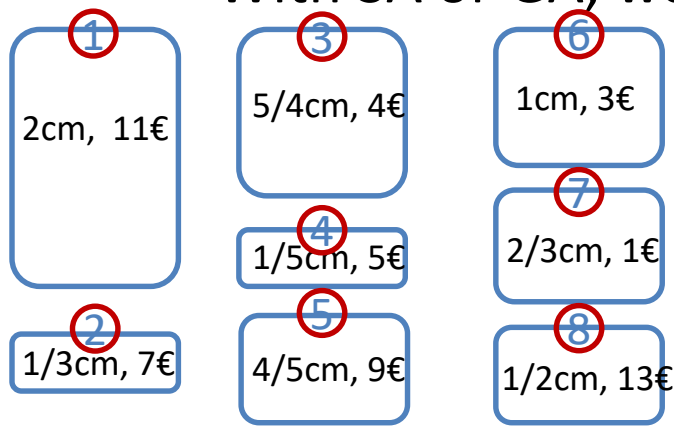
# What Makes Combinatorial Optimization Special?

- Solution consists of a **combination** of unique **components** selected from a typically finite set
- Typical example: knapsack problem
  - Given  $n$  blocks of different heights and worth different amount of money as well as a knapsack of certain height
  - Goal: Fill the knapsack with blocks worth the most money without overfilling it
- Real-world problems: Fill process queues, allocate delivery trucks, determine how much food you can serve for 13.24€



# Can't We Use Our Known Techniques?

- Yes! Iterated local search, tabu search, etc. are often used
- However, combinatorial problems are often more constrained than others (see knapsack or TSP)
- Example: Boolean vector representing whether a component is in the knapsack
  - With SA or GA, we can quickly construct infeasible solutions



Larger solutions are invalid. Should they get a zero score? Maybe, not. Use the distance to the optimum/how overfull the knapsack is.

Optimal:  $\langle false, true, false, true, true, false, true, true \rangle$

# How to Handle Invalid Solutions?

---

- TSP problem has no notion of how far away we are from a valid one
- Which quality do we assign an invalid tour?
  - Punishing a solution might work
  - Distance to closest known valid solution
  - Just removing this solution
- Hard constraints affect how we initialize our individuals and how we tweak them

# Initialization & Construction

- Idea: Iterative construction of components with hard constraints
  - Choose a component (e.g., pick an edge for TSP or initial block in knapsack) and make a partial solution out of it
  - Identify the subset of components that can be concatenated to components in our partial solution (e.g., all edges going out of cities A or B used in step one; all blocks that do not overflow the knapsack)
  - Tend to discard the less desirable components (TSP might concentrate only on those edges of unvisited cities)
  - Add to the partial solution a component from the subset
  - Quit when there are no components left or go to step 2



# Tweak Operation

- Problem: A valid solution might have only invalid neighbors
- **Closed** tweak operator
  - Mutates an individual such that we yield only a valid individual from this mutation
  - Crossover might be too challenging!
- **Repeat** tweak until valid individual has been found
  - Might take too much time!
- Allow invalid individuals via adapted fitness function
  - Adapt the fitness assessment to cope with invalid individuals (e.g., distance to nearest valid neighbor or to the optimum)
- Allow invalid individuals with a poor quality (caution: remember Hamming Cliff of exercise, because best solutions might be close to invalid ones)

# Component-Oriented Methods

---

- Idea: Solution is made of combinations of components from a fixed set
  - Use this fact by maintaining historical quality information of individual components (instead of entire individuals)
- When to apply?
  - Construction: Use components with a higher probability that have proved to be a better choice
  - Tweaking: Modify those components that have a higher chance to improve the fitness value

# Greedy Randomized Adaptive Search Procedures (GRASP)

# GRASP

- Invented by T. Feo and M. Resende in 1989: A probabilistic heuristic for a computationally difficult set covering problem
- Single-state metaheuristic based on closure tweak and construction -> we stay in the valid solution area
- No notion about component-level historical quality
- Idea: Construct valid solution based on highest value (or lowest cost) components and then do hill climbing

# GRASP Algorithm


$C \leftarrow \{C_1, C_2, \dots, C_n\}$  components

$p \leftarrow$  percentage of components to include each iteration

$m \leftarrow$  length of time to do hill climbing


$Best \leftarrow empty$

**repeat**

$S \leftarrow \{\}$   Single state, keeping the best solution of current hill-climbing run


**repeat**

$C' \leftarrow$  components in  $C - S$ , which could be added to  $S$ , without being invalid

**if**  $C'$  is *empty* **then**  Candidates that are not already in the solution and would not make the solution invalid  
 $S \leftarrow \{\}$

**else**  Start all over again, because incomplete and no valid component left

$C'' \leftarrow$  the  $p\%$  highest value (or lowest cost) components in  $C'$

$S \leftarrow S \cup \{\text{component chosen uniformly at random from } C''\}$   Sort the components based on their gain and chose one randomly; could be replaced by tournament selection or proportionate selection


**until**  $S$  is a complete solution

**for**  $m$  times **do**

$R \leftarrow \text{Tweak}(\text{Copy}(S))$

**if**  $\text{Quality}(R) > \text{Quality}(S)$  **then**

$S \leftarrow R$

 Do hill climbing on the current solution

**if**  $Best$  is *empty* or  $\text{Quality}(S) > \text{Quality}(Best)$  **then**

$Best \leftarrow S$

**until**  $Best$  is optimum or out of time

**return**  $Best$

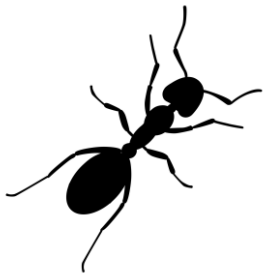
# Ant Colony Optimization (ACO)



# History



- First introduction by Marco Dorigo in 1992 as his PhD thesis
  - One of the founders of the swarm intelligence field
  - Received several awards



“Ant colony optimization studies artificial systems that take inspiration from the behavior of real ant colonies and which are used to solve discrete optimization problems.”

# Observations of Real Ants



- Limited individual capabilities
  - Rudimentary sight and auditory communication
- Sophisticated swarm capabilities
  - Regulating nest temperature within 1° Celsius range
  - Forming bridges
  - Raiding food areas
  - Building and protecting the nest
  - Carrying large items in groups
  - Emigration of the whole colony
  - Route planning and **optimization**



shortest route to food source



# Basic Principle: Stigmergy

- First used by Pierre-Paul Grasse in 1959 to describe termite behavior



- Basic idea:
  - Individuals **exchange information** (i.e., interact with each other) **via** (modifications of) **the environment**
  - Form of **self-organization** that can produce complex, purposeful structures and behavior **without the need for planning or direct communication**

# Ants and Stigmergy

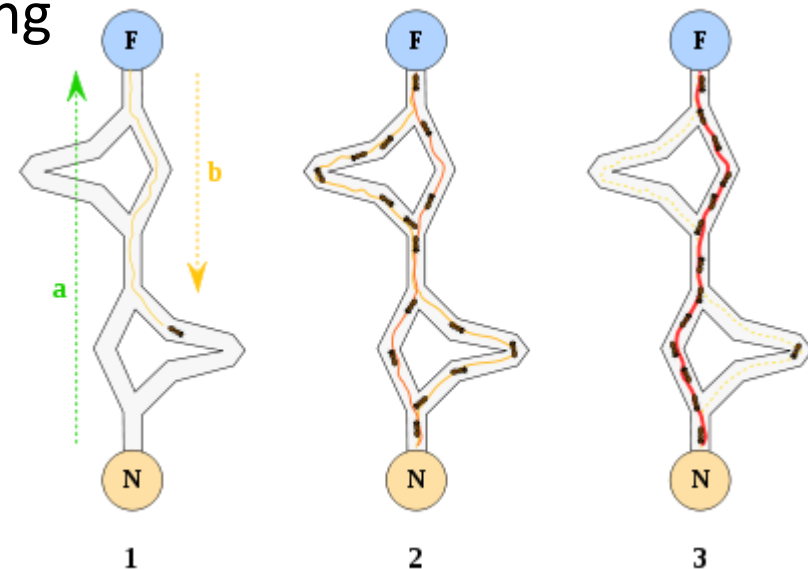
- Ants use chemicals (pheromones) on the ground to lay a trail for the following ants
- Ants will choose to follow trails with higher pheromone levels with a **higher probability** than trails with lower pheromone levels
- As more ants use a path, the pheromone trail grows stronger
- This results in an **autocatalytic behavior**, as more ants follow the trail

Positive feedback loop is main idea behind ant colony optimization



# The Double Bridge Experiment

- Experiment by Deneubourg et al. 1990 investigated the ants' ability of finding the shortest path to a food source
  - *Iridomyrmex humilis*
  - A double bridge with two branches of different size connected the nest to a food source
  - The long branch was twice as long
- Results
  - Random walk of ants at the beginning
  - After a few minutes all ants were on the short branch



# Back to Software: Properties of ACO

---

- Tweaking is optional! So, we avoid this problem
- Idea: Assembling of candidate solutions based on components that compete against each other
- Populations:
  - Set of components that make up a candidate solution
    - Fixed set
    - But, fitness of an individual component can change (the pheromone)
  - Set of candidate solutions, called ant trails
    - Each solution is built by selecting components one by one, partly based on their pheromones

# ACO Approach

- Build one or more trails
- Assess the fitness of each trail
  - Update pheromone of trail based on portions of the trail's fitness

$C \leftarrow \{C_1, C_2, \dots, C_n\}$  components

$popsiz$   $\leftarrow$  number of trails to build at once (i. e. candidate solutions)

$\vec{p} \leftarrow \langle p_1, p_2, \dots, p_n \rangle$  pheromones of the components (0 at the start)

$Best \leftarrow empty$

**repeat**

$P \leftarrow popsiz$  trails built by iteratively selecting components based on pheromones  
and cost or values

**for**  $P_i \in P$  **do**

$P_i \leftarrow (Optionally)HillClimb(P_i)$

**if**  $Best$  is empty or  $Fitness(P_i) > Fitness(Best)$  **then**

$Best \leftarrow P_i$

Update  $\vec{p}$  for components based on the fitness results for each  $P_i \in P$  in which they participated

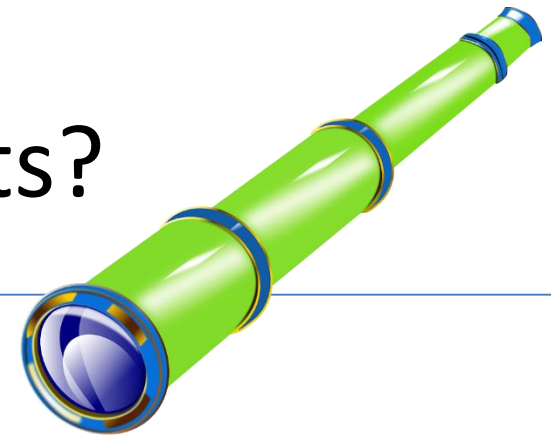
**until**  $Best$  is optimum or out of time

**return**  $Best$

# Comparing GRASP with ACO

- Both build iteratively candidate solutions and perform hill climbing on them
- Differences:
  - ACO builds all *popsizes* solutions at once
  - Hill climbing in ACO is optional, so no closed tweak operation required
  - Component selection bases not only on their value, but also on pheromones, which are the historical quality of a component (e.g., mean fitness value of all trails in which so component occurs)

# Where are the Ants?



- There are no ants!
- ACO is inspired by previous work on pheromone-based ant foraging and trail formation
- What is there?
  - Components with historical information about quality
  - Candidate solutions formed from those components with fitness values, which are mapped to their components

# Concrete Realization: The Ant System (AS)

---

- First version of ACO
- Component selection based on fitness-proportionate selection using cost/value and pheromones
- Fitness values are added to the component pheromones
- To control growth of pheromone values, we reduce (evaporate) all pheromones at each generation
- Algorithm:
  - Construct some trails by selecting components
  - (Optionally) do hill climbing on those
  - Assess the fitness of all trails
  - Evaporate all pheromones by a certain amount
  - Update the pheromones that participate in trails based on the respective fitness value



# Ant System Algorithm I

---

## Initialization

$C \leftarrow \{C_1, C_2, \dots, C_n\}$  components

$popsiz$   $\leftarrow$  number of trails to build at once (i. e. candidate solutions)

$e \leftarrow$  evaporation constant:  $0 < e \leq 1$






$\gamma \leftarrow$  initial value of pheromones

$t \leftarrow$  number of iterations for hill climbing

$\vec{p} \leftarrow \langle p_1, p_2, \dots, p_n \rangle$  pheromones of the components (0 at the start)

$Best \leftarrow empty$

# Ant System Algorithm II

```
repeat
   $P \leftarrow \{ \}$ 
  for  $popsiz$ e times do
     $S \leftarrow \{ \}$ 
    repeat  Create the population for each iteration from scratch
       $C' \leftarrow$  components in  $C - S$  that could be added to  $S$  without being invalid
      if  $C'$  is empty then  Could not create a valid solution -> start from scratch
         $S \leftarrow \{ \}$ 
      else
         $S \leftarrow S \cup \{ \text{component selected from } C' \text{ based on pheromones and value or cost} \}$ 
    until  $S$  is a complete trail
     $S \leftarrow HillClimb(S)$  for  $t$  iterations
    AssessFitness( $S$ )
    if  $Best$  is empty or  $Fitness(S) > Fitness(Best)$  then  Tune and assess fitness of population
       $Best \leftarrow S$ 
     $P \leftarrow P \cup \{S\}$ 
    for each  $p_i \in \vec{p}$  do  Evaporate the pheromones
       $p_i \leftarrow (1 - e) * p_i$ 
    for each  $P_j \in P$  do  Increase the pheromones by the fitness values of the solutions in which they are part of
      for each component  $C_i$  do
        if  $C_i$  was used in  $P_j$  then
           $p_i \leftarrow p_i + Fitness(P_j)$ 
    until  $Best$  is optimum or out of time
  return  $Best$ 
```

# Specific Aspects of AS

- Constructing solutions can be way more complex
- Construction is based on **desirability** of a component

$$Desirability(C_i) = p_i^\delta * (Value(C_i))^\varepsilon$$

- Where  $\delta$  and  $\varepsilon$  are tuning parameters
- Do proportionate or tournament selection on this value
- Initialization of pheromones
  - $\gamma = 1$  or  $\gamma = popsize * (\frac{1}{Cost(D)})$ , where  $D$  is a costly tour
- Evaporation of pheromones
  - To avoid piling up the fitness values
  - Alternative: Adjust the value based on how well the component has performed on average

# Update Pheromones Using Learning Rate

$C \leftarrow \{C_1, C_2, \dots, C_n\}$  components

$\vec{p} \leftarrow \langle p_1, p_2, \dots, p_n \rangle$  pheromones of the components (0 at the start)

$P \leftarrow \{P_1, P_2, \dots, P_M\}$  population (trails)

$\alpha \leftarrow$  learning rate:  $0 \leq \alpha \leq 1$

$\vec{r} \leftarrow \langle r_1, r_2, \dots, r_n \rangle$  total desirability of each component, initialized with 0

$\vec{c} \leftarrow \langle c_1, c_2, \dots, c_n \rangle$  component usage counts, initialized with 0

**for each**  $P_j \in P$  **do**

**for each** component  $C_i$  **do**

**if**  $C_i$  was used in  $P_j$  **then**

$r_i \leftarrow r_i + \text{Desirability}(P_j)$

$c_i \leftarrow c_i + 1$

**for each**  $p_i \in \vec{p}$  **do**

**if**  $c_i > 0$  **then**

$p_i \leftarrow (1 - \alpha) * p_i + \alpha * \frac{r_i}{c_i}$

**return**  $\vec{p}$

Go through each trail and each component and check whether this component is used in the trail, if so, increase the counter and update the desirability of that component

Update each pheromone based on the learning rate to reduce the value of the previous iteration and add a corresponding portion based on the average desirability of this iteration

# Concrete Realization: Ant Colony System (ACS)

- Similar to AS, but with following improvements:
  - Elitism for pheromone updates: Increase only pheromones that are part of the best trails:
    - $p_i \leftarrow (1 - \alpha) * p_i + \alpha * \text{Fitness}(\text{best})$
  - Learning rate in pheromone updates
  - Better pheromone evaporation
    - $p_i \leftarrow (1 - \beta) * p_i + \beta * \gamma$ , where  $\beta$  is an “unlearning rate” and  $\gamma$  is the initial value of a pheromone
  - Stronger emphasis on components during selection that were in the best trails before

# ACS Algorithm I

## Initialization

$C \leftarrow \{C_1, C_2, \dots, C_n\}$  components

$popsiz$   $\leftarrow$  number of trails to build at once (i. e. candidate solutions)

$\alpha \leftarrow$  elitist learning rate

$\beta \leftarrow$  evaporation rate

$\gamma \leftarrow$  initial value of pheromones

$\delta \leftarrow$  tuning parameter for heuristic in component selection

$\varepsilon \leftarrow$  tuning parameter for pheromones in component selection

$t \leftarrow$  number of iterations for hill climbing

$q \leftarrow$  probability of selecting components in an elitist way

$\vec{p} \leftarrow \langle p_1, p_2, \dots, p_n \rangle$  pheromones of the components (0 at the start)

$Best \leftarrow empty$

# ACS Algorithm II

```
repeat
   $P \leftarrow \{ \}$ 
  for  $popsiz$ e times do
     $S \leftarrow \{ \}$ 
    repeat
       $C' \leftarrow$  components in  $C - S$  that could be added to  $S$  without being invalid
      if  $C'$  is empty then
         $S \leftarrow \{ \}$ 
      else
         $S \leftarrow S \cup \{\text{component selected from } C' \text{ using } \textit{elitist component selection}\}$ 
    until  $S$  is a complete trail
     $S \leftarrow HillClimb(S)$  for  $t$  iterations
    AssessFitness( $S$ )
    if  $Best$  is empty or  $Fitness(S) > Fitness(Best)$  then
       $Best \leftarrow S$ 
      Same code as in AS
  for each  $p_i \in \vec{p}$  do
     $p_i \leftarrow (1 - \beta) * p_i + \beta * \gamma$   $\leftarrow$  Decrease all pheromones using a certain rate
  for each component  $C_i$  do
    if  $C_i$  was used in  $Best$  then
       $p_i \leftarrow (1 - \alpha) * p_i + \alpha * Fitness(Best)$  } Increase pheromones only for the
      components that are used in the best
    solution
until  $Best$  is optimum or out of time
return  $Best$ 
```

# Comparing ACO with ACS

- Ant System:
  - Selection of components is greedily based on how often a component was in a high-quality/best solution and how well this solution was rated
  - Ignores that a component might be **linked** with another one to be really good
- Ant Colony System
  - Circumvents this problem as it gives all components in a good solution a large amount of pheromones
  - Linkage will be considered as in coevolution
  - Think about pair-wise or higher-order pheromones that get reward if a pair or more components appear together in a good solution (hint: might be a solution for the project!)



# Guided Local Search (GLS)

# Guided Local Search (GLS)

- Idea: Mark components that participate in local optima solutions and try to **not** select them further
- Recall: Tabu Search maintained a list of solutions that we avoid visiting/using in the future
- Recall: Feature-Based Tabu Search finds features in a solution that lead to good solutions and make them taboo for certain time steps
- Consider components as features and having a closed tweak operator, we can just apply Feature-Based Tabu Search for combinatorial optimization

# GLS is a Variant of Feature-Based TS

---

- Developed by C. Voudouris and E. Tsang in 1995
- Similar to ACO, assign historical quality to the components
- But, usage is different:
  - Here, we use the information to make problematic components taboo and force exploration
- At the end, GLS is hill climbing on components, where components are penalized that appear too often in local optima

# Realization of GLS

- Maintain a vector of pheromones, counting the appearance of the corresponding component in good solutions
- Hill climb not on quality, but on an *adjusted quality function*
  - Consider quality **and** pheromones of the used components

$$AdjustedQuality(S, C, \vec{p}) = Quality(S) - \beta \sum_i \begin{cases} p_i & \text{if } C_i \in S \\ 0 & \text{otherwise} \end{cases}$$

Where  $S$  is a candidate solution,  $C$  the set of components, and  $\vec{p}$  the vector of pheromones

- Result: Look at high quality and novel solutions
  - $\beta$  parameter controls the importance of novelty

# How to Adjust Pheromones?

- Pheromones of good solutions will be increased, but only under the following conditions:
  - Appearance in the good/best solution (idea: go away from local optima)
  - Component has lower value/higher cost compared to others (idea: go away from the least important components first)
  - Pheromone is not already high (idea: do not raise the amount to a level where the corresponding component can never be taken again)

$$Penalizability(C_i, p_i) = \frac{1}{(1 + p_i) * Value(C_i)} = \frac{Cost(C_i)}{(1 + p_i)}$$

# How to Penalize?

---

- Pick the most penalizable component of the current solution and increment its pheromone by 1
- Somehow similar to desirability function: most penalized components are the least desirable
- Note: There is no evaporation

# GLS Algorithm

$C \leftarrow \{C_1, C_2, \dots, C_n\}$  components  
 $T \leftarrow$  distribution of possible time intervals  
 $S \leftarrow$  initial candidate solution  
 $\vec{p} \leftarrow \langle p_1, p_2, \dots, p_n \rangle$  pheromones of the components (0 at the start)  
 $Best \leftarrow \text{empty}$

**repeat**

$time \leftarrow$  random time in the near future, chosen from  $T$

**repeat**

$R \leftarrow \text{Tweak}(\text{Copy}(S))$

**if**  $\text{Quality}(R) > \text{Quality}(Best)$  **then**

$Best \leftarrow R$

**if**  $\text{AdjustedQuality}(R, C, \vec{p}) > \text{AdjustedQuality}(S, C, \vec{p})$  **then**

$S \leftarrow R$

**until**  $Best$  is optimal or  $time$  is over or out of time

$C' \leftarrow \{ \}$

**for** each component  $C_i \in C$  appearing in  $S$  **do** ← Find most penalizable components

**if** for all  $C_j \in C$  appearing in  $S$ ,  $\text{Penalizability}(C_i, p_i) \geq \text{Penalizability}(C_j, p_j)$  **then**

$C' \leftarrow C' \cup \{C_i\}$

**for** each component  $C_i \in C$  appearing in  $S$  **do**

**if**  $C_i \in C'$  **then**

$p_i \leftarrow p_i + 1$

Penalize them by increasing  
their pheromones

Do hill climbing using  
pheromone-adjusted  
quality measure

**until**  $Best$  is optimal or out of time

**return**  $Best$

# Take Home Message:

---

- Combinatorial optimization is a special kind of optimization, where we want to find the best combination of a fixed set of components
- Ideas are that we keep track of historical quality of components
- Ant-based approaches try to favor the selection of components that have appeared in high performing solutions -> exploitation
- Guided local search tries to disfavor the selection of components that have appeared in high performing solutions -> exploration
- Have you thought about Guided Genetic Algorithm?



# Next Lecture

---

- Constraint Satisfaction Problem Solving
  - How can we construct solutions when having constraints?
  - What are approaches to find valid solutions faster?
- Goal: Plug in such solutions into initialization or tweak procedures of approaches that we have seen so far

# References

- Search Methodologies: Introductory Tutorials in Optimization and Decision by Edmund K. Burke, Graham Kendall, 2013
- Self-organising Software: From Natural to Artificial Adaptation by Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, Anthony Karageorgos, 2011

