Heuristic Optimization Methods

VNS, GLS

Variable Neighborhood Search (1)

- Based on the following central observations
  - A local minimum w.r.t. one neighborhood structure is not necessarily locally minimal w.r.t. another neighborhood structure
  - A global minimum is locally optimal w.r.t. *all* neighborhood structures
  - For many problems, local minima with respect to one or several neighborhoods are relatively close to each other

Variable Neighborhood Search (2)

- Basic principle: change the neighborhood during the search
- Variations:
  - Variable Neighborhood Descent
  - Basic Variable Neighborhood Search
  - Reduced Variable Neighborhood Search
  - Variable Neighborhood Decomposition Search

Variable Neighborhood Search (3)

- The first task is to generate the different neighborhood structures
- For many problems different neighborhood structures already exists
  - E.g., for the VRP: 2-Opt, Cross, Swap, Exchange,…
- Find neighborhoods that depend on some parameter
  - k-Opt (k=2,3,…)
  - Flip-neighborhoods can be extended to double-flip, triple-flip, etc…
- Some neighborhoods are associated with distance measures: can increase the distance

Variable Neighborhood Descent

1: input: starting solution, s_0
2: input: neighborhood operators, $\{N_k\}$, $k = 1, \ldots, k_{max}$
3: input: evaluation function, f
4: $current \Leftarrow s_0$
5: $k \Leftarrow 1$;
6: **while** $k \leq k_{max}$ **do**
7: $s \Leftarrow$ the best neighbor in $N_k(current)$
8: **if** $f(s) < f(current)$ **then**
9: $current \Leftarrow s$
10: $k \Leftarrow 1$
11: **else**
12: $k \Leftarrow k + 1$
13: **end if**
14: **end while**

Variable Neighborhood Descent

- The final solution is locally optimal with respect to all neighborhoods, N_1, N_2,…, $N_{k\text{-}max}$
- ”First Improvement” could be used instead of ”Best Improvement”
- Typically, neighborhoods are ordered from smallest to largest
- If Local Search Algorithms can be treated as Black-Box procedures:
  - Sort the procedures
  - Apply them in the given order
  - Possibly reiterate starting the first one
  - Advantage: solution quality and speed

Basic Variable Neighborhood Search

- Use neighborhood structures N_1, N_2,…, $N_{k\text{-}max}$
- Standard (”Best Improvement”) Local Search is applied in N_1
- The other neighborhoods are explored only randomly
- Exploration of the other neighborhoods are perturbations as in ILS
  - Perturbation is systematically varied

Basic Variable Neighborhood Search

1: input: starting solution, s_0
2: input: neighborhood operators, $\{N_k\}$, $k = 1, \ldots, k_{max}$
3: input: Local Search procedure LS, using a different neighborhood operator
4: input: evaluation function, f
5: $current \Leftarrow s_0$
6: **while** stopping criterion not met **do**
7: $k \Leftarrow 1$;
8: **while** $k \leq k_{max}$ **do**
9: $s \Leftarrow$ a random solution in $N_k(current)$
10: $s* \Leftarrow LS(s)$
11: **if** $f(s*) < f(current)$ **then**
12: $current \Leftarrow s*$
13: $k \Leftarrow 1$
14: **else**
15: $k \Leftarrow k + 1$
16: **end if**
17: **end while**
18: **end while**

Variations of Basic VNS

- The order of the neighborhoods
  - Forward VNS: start with k=1 and increase
  - Backward VNS: start with $k=k_{max}$ and decrease
  - Extended version:
    - Parameters k_{min} and k_{step}
    - Set $k = k_{min}$, and increase k by k_{step} if no improvement
- Accepting worse solutions
  - With some probability
  - Skewed VNS: Accept if
    - $f(s^*)-\alpha d(s, s^*) < f(s)$
    - $d(s, s^*)$ measures the distance between the solutions

Final Notes on VNS (1)

- Other variations exists
  - Reduced VNS: same as Basic VNS, but no Local Search procedure
    - Can be fast
  - Variable Neighborhood Decomposition Search
    - Fix some components of the solution, and perform Local Search on the remaining ”free” components

Final Notes on VNS (2)

- ILS and VNS are based on different underlying ”philosophies”
  - ILS: Perturb and do Local Search
  - VNS: Exploit different neighborhoods
- ILS and VNS are also similar in many respects
- ILS can be more flexible w.r.t. the optimization of the interaction between modules
- VNS gives place to approaches such as VND for obtaining more powerful Local Search approaches

Conclusions about ILS and VNS

- Based on simple principles
- Easy to understand
- Basic versions are easy to implement
- Robust
- Highly effective

Guided Local Search

- A general strategy, metaheuristic, used to guide a neighborhood search
- Tries to overcome local optima by ”removing” them:
  - Changes the ”topography” of the search space
  - Uses an extended move evaluation function
    - Original objective function + penalties
- Focuses on promising parts of the search space

Features of a Solution

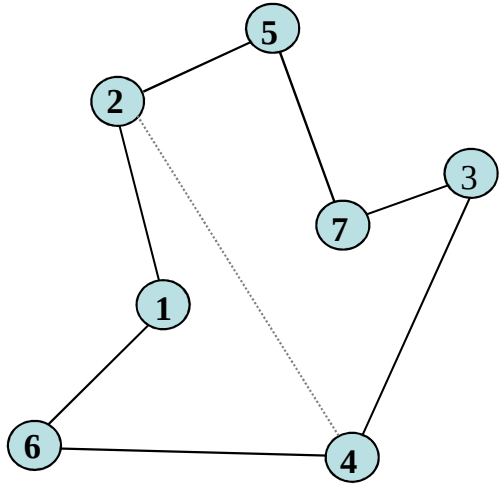
- GLS assumes that we can find some features of a solution that we can penalize
- What is a ”feature”?
  - Something that is characteristic for the solution
  - Something that might be different in another solution
  - Problem dependent
- Examples:
  - TSP: whether city A follows immediately after city B in the tour or not
  - Knapsack: whether item 5 is included or not

Features - Example: TSP

- A solution (trip) is an ordered set of edges
- An edge is a good choice for feature:
  - Is either in a solution or not
  - Feature cost = edge length
- Let the set of all edges e_{ij} be features:

$$E = \{e_{ij}\},\ i = 1 \ldots N,\ j = i+1 \ldots N,\ i \neq j$$

- The cost for a feature e_{ij} is given by d_{ij} in the distance matrix:



$$\mathbf{D} = [d_{ij}],\ i = 1 \ldots N,\ j = 1 \ldots N$$

Features & GLS

- The modification of the move evaluation in GLS is based on features
- The features each has a cost
  - Represents (directly or indirectly) the influence of a solution on the (extended) move evaluation function
  - Constant or variable (dependent on other features)
  - GLS tries to avoid costly features
- We use an indicator function as follows:
  - $I_i(s) = 1$ if solution s has feature i
  - $I_i(s) = 0$ if solution s does not have feature i

Extended Move Evaluation (1)

- Until now we have only seen the use of the objective function in move evaluations
- We let f be the objective function
  - *f(s)* gives us the value of a solution *s*
- We have always taken the best neighbor to be the neighbor *s* for which *f(s)* has the best value
- This will make the search stop when a local optimum have been found

Extended Move Evaluation (2)

- Let the set of features be denoted by
  - F = {1,…,G}
- We have our indicator function:
  - $I_i(s)$ = 1 if solution *s* has feature *i*
  - $I_i(s)$ = 0 if solution *s* does not have feature *i*
- We create a penalty vector $\mathbf{p}=[p_i]$, $i=1\ldots G$
  - p_i is the number of times feature *i* have been penalized until now
- The extended move evaluation function becomes
  - $$f^*(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

Extended Move Evaluation (3)

- The extended move evaluation function has two parts
  - The original objective function
  - A penalty term, which penalizes certain *features* of the solution

$$f^*(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

- The parameter λ adjusts the influence of the penalties

Penalties (1)

$$f^*(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

- The penalties are initially equal to 0
- When the search has reached a local optimum (with respect to the extended move evaluation function)
  - The penalty is increased for some of the features of the current (locally optimal) solution
  - This will make the current solution look worse in comparison to some neighboring solutions (which do not have the penalized features, and thus do not get the penalty)

Penalties (2)

- How to select which feature to penalize?
- Define the *utility* of a feature *i* in solution *s* as follows:
  - $u_i(s) = I_i(s) * c_i / (1+p_i)$
  - Here, c_i is the cost of the feature (in objective function) and p_i is the previous penalty
- In a local optimum, *s*, increase the penalty for the feature that has the highest utility value, $u_i(s)$
- NB: Penalties are only adjusted when the search has reached a local optimum, and only for features included in the local optimum

Guided Local Search

1: input: starting solution, s_0
2: input: neighborhood operator, N
3: input: evaluation function, f
4: input: a set of features, F
5: input: a penalty factor, λ
6: $current \Leftarrow s_0$
7: $best \Leftarrow s_0$
8: $p_i \Leftarrow 0$ (for all $i \in F$)
9: **while** stopping criterion not met **do**
10: Define $f^*(s) = f(s) + \lambda \sum_{i \in F} I_i(s) p_i$
11: $s^* \Leftarrow$ the best solution in $N(current)$, according to f^*
12: **if** $f^*(s^*) < f^*(current)$ **then**
13: $current \Leftarrow s^*$
14: **if** $f(current) < f(best)$ **then**
15: $best \Leftarrow current$
16: **end if**
17: **else**
18: Define the utility, $u_i(current) = I_i(current) \frac{c_i}{1+p_i}$, for all $i \in F$
19: $p_i \Leftarrow p_i + 1$ for each feature $i \in F$ having the maximum utility in solution $current$
20: **end if**
21: **end while**

Comments on GLS

- Uses the extended move evaluation function when deciding which neighbor is best
- Could also use ”First Improvement”-strategy
- Uses the normal objective function value to find the best solution
  - **if** ($f(current)<f(best)$ …
- If all features er penalized equally, then $f^*(s)$ describes the same ”landscape” as $f(s)$

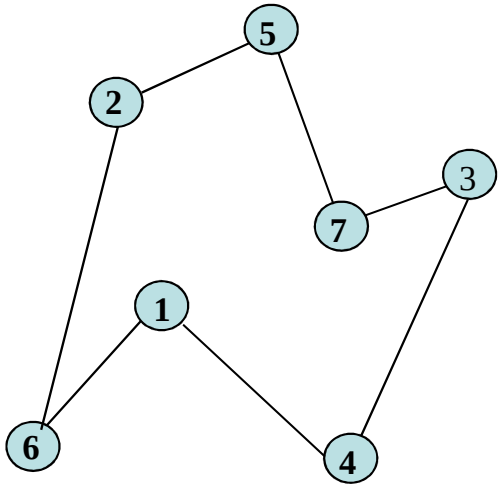
How to Select Lambda

$$f^*(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

- The control parameter λ dictates the influence of the penalty on the extended move evaluation function
  - Low value: intensification
  - High value: diversification
- Can be problematic to find values for λ, even though the method is robust for some value
- Generally: fraction of the objective function value at a local minimum
  - λ = α**f*(a local minimum)/(#features in the local minimum)

GLS - Example : TSP (1)

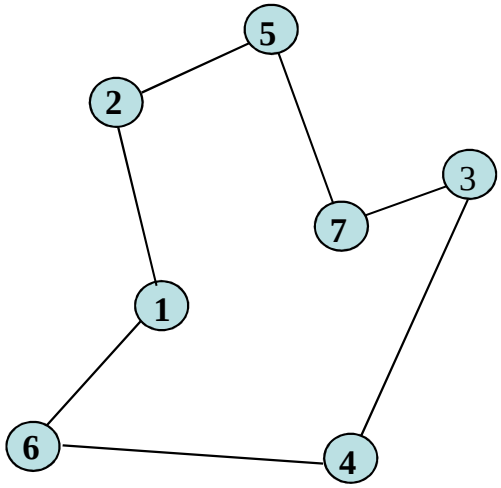
- Features: edges
- Cost of the features: edge length
- The feature associated with e_{26} will be penalized in the solution on the right:
  - In the next round of LS is the move evaluation function as before, *f(s)*, except if e_{26} is in the solution, when the value will be $f(s)+\lambda$



	1	2	3	4	5	6	7
1		0	0	0	0	0	0
2			0	0	0	1	0
3				0	0	0	0
4					0	0	0
5						0	0
6							0

GLS - Example : TSP (2)

- After the next LS e_{34} is penalized
- After this the move evaluation function is as before, *f(s)*, except if e_{26} or e_{34} is in the solution



	1	2	3	4	5	6	7
1		0	0	0	0	0	0
2			0	0	0	1	0
3				1	0	0	0
4					0	0	0
5						0	0
6							0

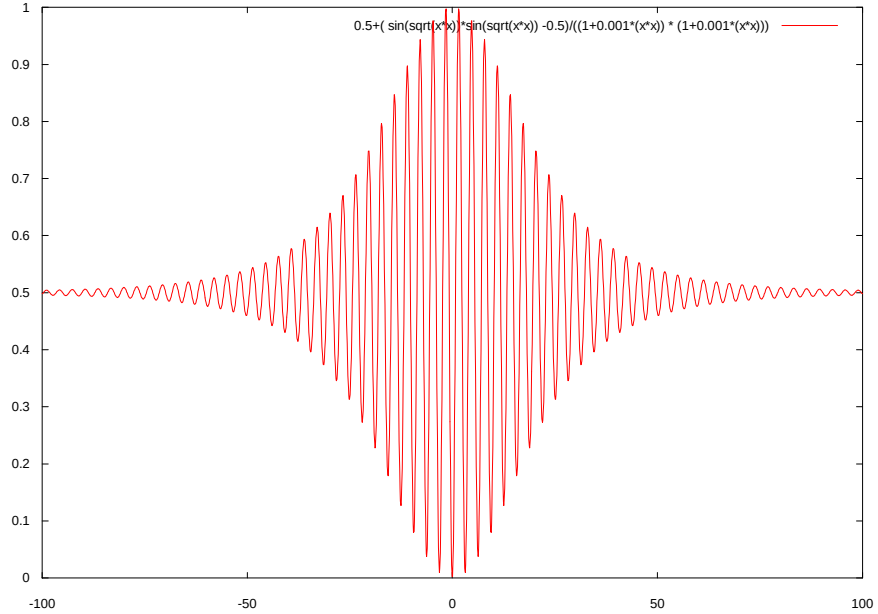
$$f^*(s) = f(s) + \begin{cases} 0 & e_{26} \text{ and } e_{34} \text{ not used in } s \\ \lambda & \text{one of } e_{26} \text{ and } e_{34} \text{ used in } s \\ 2\lambda & e_{26} \text{ and } e_{34} \text{ both used in } s \end{cases}$$

Applications of GLS

- A fairly recent metaheuristic (evolved from another method called GENET)
- Some applications:
  - Workforce scheduling at BT (Tsang, Voudouris 97)
  - TSP (Voudouris, Tsang 98)
  - VRP (Kilby et. al. 97)
  - Function optimization (Voudouris 98)
  - …

Possibilities and Extensions

- Limited life time of penalties
- Diminishing penalties
- Awards in addition to penalties
- Automatic regulation of λ
- New utility-functions to find features to penalize



- Has been used for function optimization, good results on:

$$F6(x,y) = 0.5 + \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{\left[1+0.001(x^2+y^2)\right]^2}$$

GLS versus SA

- It is difficult in SA to find the right cooling schedule (problem dependent)
  - High temperature gives bad solutions
  - Low temperature gives convergence to a local minimum
  - SA is non-deterministic
- GLS visits local minima, but can escape
  - Not random up-hill moves as in SA
  - GLS is deterministic
  - Does not converge to a local minimum; penalties are added until the search escapes

GLS vs. Tabu Search (1)

- GLS is often said to be closely related to Tabu Search
- Both have mechanisms to guide the Local Search away from local optima
  - GLS penalizes features in the solutions
  - TS bans (makes taboo) features in the solutions
- Both incorporate memory structures
  - GLS has the accumulated penalties
  - TS has different memory structures
    - Short term, long term, frequency, recency, …

GLS vs. Tabu Search (2)

	Tabu Search	**GLS**
Information that is used	Modified neighborhood	Modified move evaluation function
Memory	Tabu list, frequency based memory	Penalizing features
When?	Every iteration or every N'th iteration	In local minima defined by the extended move evaluation function
The "nature" of the search	- Avoid stops in local minima and reversing moves - Diversification; penalize moves that are done often or attributes appearing often in solutions	- Escape local minima - Distribute the search effort base don the cost of features
Intensification / diversification	< >	Parameter λ
Reduction of neighborhood	Candidate list	Fast Local Search - FLS

Summary

- VNS
  - Idea: utilize different neighborhoods
  - Several variations
  - Variable Neighborhood Descent
- GLS
  - Idea: Remove local optima by changing the evaluation of solutions
  - Has some similarities with Tabu Search