DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS
FOR COMBINATORIAL OPTIMZATION

Lecture 12

Stochastic Local Search Methods

Marco Chiarandini

Outline

Outline

1. Stochastic Local Search Methods (Metaheuristics)
   Randomized Iterative Improvement
   Attribute Based Hill Climber
   Dynamic Local Search
   Iterated Local Search
   Tabu Search

‘Simple’ SLS Methods

Goal:

Effectively escape from local minima of given evaluation function.

General approach:

For fixed neighborhood, use step function that permits
worsening search steps.

Specific methods:

- Randomized Iterative Improvement
- (Simulated Annealing)
- Attribute Based Hill Climber
- Dynamic Local Search
- Iterated Local Search
- Tabu Search

Min-Conflict Heuristics

procedure *WalkSAT* (*F*, *maxTries*, *maxSteps*, *slc*)
  input: *CNF formula F, positive integers maxTries and maxSteps, heuristic function slc*
  output: *model of F* **or** 'no solution found'
  for *try* := 1 **to** *maxTries* **do**
    a := randomly chosen assignment of the variables in formula *F*;
    for *step* := 1 **to** *maxSteps* **do**
      if *a* satisfies *F* **then return** *a* **end**
      c := randomly selected clause unsatisfied under *a*;
      x := variable selected from *c* according to heuristic function *slc*;
      a := *a* with *x* flipped;
    end
  end
  return 'no solution found'
end *WalkSAT*

Figure 6.3 The WalkSAT algorithm family. All random selections are according to a uniform probability distribution over the underlying sets; WalkSAT algorithms differ in the variable selection heuristic *slc*.

Randomized Iterative Improvement

Key idea: In each search step, with a fixed probability perform an uninformed random walk step instead of an iterative improvement step.

Randomized Iterative Improvement (RII):
determine initial candidate solution s
while termination condition is not satisfied **do**
  With probability *wp*:
    choose a neighbor s' of s uniformly at random
  Otherwise:
    choose a neighbor s' of s such that $g(s') < g(s)$ or,
     if no such s' exists, choose s' such that $g(s')$ is minimal
  $s := s'$

Example: Randomized Iterative Improvement for GCP

procedure *RIIGCP*$(F, wp, maxSteps)$
  input: *a graph* G *and* k, *probability wp, integer maxSteps*
  output: *a proper coloring* φ *for* G **or** $\emptyset$
  choose coloring φ of G uniformly at random;
  steps := 0;
  while not(φ is not proper) **and** (*steps* < *maxSteps*) **do**
    with probability *wp* **do**
      select v in V and c in Γ uniformly at random;
    otherwise
      select v in V^c and c in Γ uniformly at random from those that
        maximally decrease number of edge violations;
    change color of v in φ;
    steps := *steps*+1;
  end
  if φ is proper for G **then return** φ
  else return $\emptyset$
  end
end *RIIGCP*

Note:

- No need to terminate search when local minimum is encountered

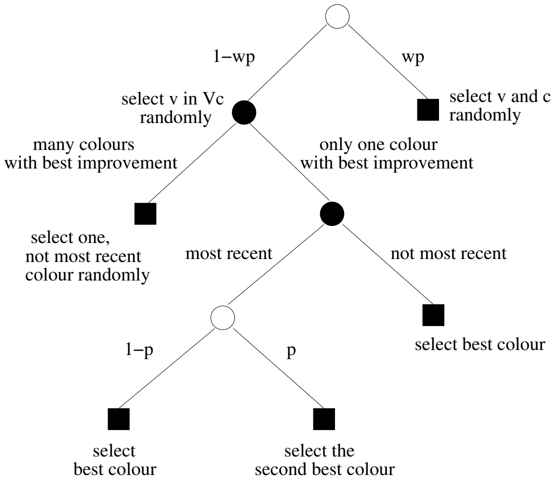
  Instead: Impose limit on number of search steps or CPU time, from beginning of search or after last improvement.
- Probabilistic mechanism permits arbitrary long sequences of random walk steps

  Therefore: When run sufficiently long, RII is guaranteed to find (optimal) solution to any problem instance with arbitrarily high probability.
- A variant of RII has successfully been applied to SAT (GWSAT algorithm)
- A variant of GUWSAT, GWSAT [Selman et al., 1994], was at some point state-of-the-art for SAT.
- Generally, RII is often outperformed by more complex LS methods.

Novelty

Key idea: combine Randomized Iterative Improvement with Min-Conflicts

Example on GCP



Attribute Based Hill Climber

- attributes are solution elements that change in a move
- each attribute has associated a value:
  - the value of the best solution visited that contains it
  - infinity, otherwise
- at each step, a solution in N is acceptable iff it contains an attribute that has never been seen in a solution of such high quality before

$$N'(s) = \{s' \in N(s) \ : \ \exists a \in s' \text{ s.t. } f(s') < \phi(a)\}$$

where

$$\phi(a) = \begin{cases} \infty & \text{if } V \cap S^a = \emptyset \\ \min\{f(s) : s \in V \cap S^a\} & \text{otherwise} \end{cases}$$

with V set of visited solutions and S^a set of solutions that contains a.

Examples

- TSP:

$$A_{TSP} = \{(i,j) \mid (i,j) \in E\}$$

- QAP:

$$A_{QAP} = \{(i,j) \mid 1 \le i \le n, 1 \le j \le n\} \text{ representing } (\varphi(i), j)$$

ALGORITHM 1. accepts(s, t)

if $\exists a \in (t - s)$ s.t. $f(t) <$ amem$[a]$ **then**
  return true
else if $f(t) \geqslant f(s)$ **then**
  return false
else
  for $i = 1$ to $|s - t| + 1$ **do**
    $a \leftarrow$ worst(i, s)
    if $f(t) \geqslant$ amem$[a]$ **then**
      return false
    else if $a \notin (s - t)$ **then**
      return true
    end if
  end for
end if

Dynamic Local Search

- **Key Idea:** Modify the evaluation function whenever a local optimum is encountered.
- Associate *weights* (*penalties*) with solution components; these determine impact of components on evaluation function value.
- Perform Iterative Improvement; when in local minimum, increase penalties of some solution components until improving steps become available.

Dynamic Local Search (DLS):
determine *initial candidate solution* s
initialize penalties
while *termination criterion* is not satisfied **do**
  compute *modified evaluation function* g' from g
    based on *penalties*
  perform *subsidiary local search* on s
    using *evaluation function* g'
  update penalties based on s

Dynamic Local Search (continued)

- **Modified evaluation function:**

$$g'(\pi, s) := g(\pi, s) + \sum_{i \in SC(\pi', s)} \mathtt{penalty}(i),$$

  where $SC(\pi', s)$ is the set of solution components of problem instance π' used in candidate solution s.

- **Penalty initialization:** For all i: $\mathtt{penalty}(i) := 0$.
- **Penalty update** in local minimum s: Typically involves *penalty increase* of some or all solution components of s; often also occasional *penalty decrease* or *penalty smoothing*.
- **Subsidiary local search:** Often *Iterative Improvement*.

Potential problem:

Solution components required for (optimal) solution may also be present in many local minima.

Possible solutions:

A: Occasional decreases/smoothing of penalties.

B: Only increase penalties of solution components that are least likely to occur in (optimal) solutions.

Implementation of B:

Only increase penalties of solution components i with maximal utility [Voudouris and Tsang, 1995]:

$$\mathtt{util}(s', i) := \frac{g_i(\pi, s')}{1 + \mathtt{penalty}(i)}$$

where $g_i(\pi, s')$ is the solution quality contribution of i in s'.

Example: Guided Local Search (GLS) for the TSP

[Voudouris and Tsang 1995; 1999]

- **Given:** TSP instance G
- **Search space:** Hamiltonian cycles in G with n vertices;
- **Neighborhood:** 2-edge-exchange;
- **Solution components** edges of G;
  $g_e(G, p) := w(e)$;
- **Penalty initialization:** Set all edge penalties to zero.
- **Subsidiary local search:** Iterative First Improvement.
- **Penalty update:** Increment penalties for all edges with maximal utility by

$$\lambda := 0.3 \cdot \frac{w(s_{\textit{2-opt}})}{n}$$

  where $s_{\textit{2-opt}}$ = 2-optimal tour.

Hybrid Methods

Combination of 'simple' methods often yields
substantial performance improvements.

Simple examples:

- Commonly used restart mechanisms can be seen as hybridisations with Uninformed Random Picking
- Iterative Improvement + Uninformed Random Walk = Randomized Iterative Improvement

Iterated Local Search

Key Idea: Use two types of LS steps:

- *subsidiary local (local) search* steps for reaching local optima as efficiently as possible (intensification)
- *perturbation steps* for effectively escaping from local optima (diversification).

Also: Use *acceptance criterion* to control diversification *vs* intensification behavior.

Iterated Local Search (ILS):
determine initial candidate solution s
perform *subsidiary local search* on s
while termination criterion is not satisfied **do**
  $r := s$
  perform *perturbation* on s
  perform *subsidiary local search* on s
  based on *acceptance criterion*,
    keep s or revert to $s := r$

Note:

- *Subsidiary local search* results in a local minimum.
- ILS trajectories can be seen as walks in the space of local minima of the given evaluation function.
- *Perturbation phase* and *acceptance criterion* may use aspects of *search history* (*i.e.*, limited memory).
- In a high-performance ILS algorithm, *subsidiary local search*, *perturbation mechanism* and *acceptance criterion* need to complement each other well.

Subsidiary local search:

- More effective subsidiary local search procedures lead to better ILS performance.
  Example: 2-opt *vs* 3-opt *vs* LK for TSP.
- Often, subsidiary local search = iterative improvement, but more sophisticated LS methods can be used. (*e.g.*, Tabu Search).

Perturbation mechanism:

- Needs to be chosen such that its effect *cannot* be easily undone by subsequent local search phase.
  (Often achieved by search steps larger neighborhood.)
  Example: local search = 3-opt, perturbation = 4-exchange steps in ILS for TSP.
- A perturbation phase may consist of one or more perturbation steps.
- Weak perturbation ⇒ short subsequent local search phase; *but:* risk of revisiting current local minimum.
- Strong perturbation ⇒ more effective escape from local minima; *but:* may have similar drawbacks as random restart.
- Advanced ILS algorithms may change nature and/or strength of perturbation adaptively during search.

Acceptance criteria:

- Always accept the best of the two candidate solutions

  ⇒ ILS performs Iterative Improvement in the space of local optima reached by subsidiary local search.

- Always accept the most recent of the two candidate solutions

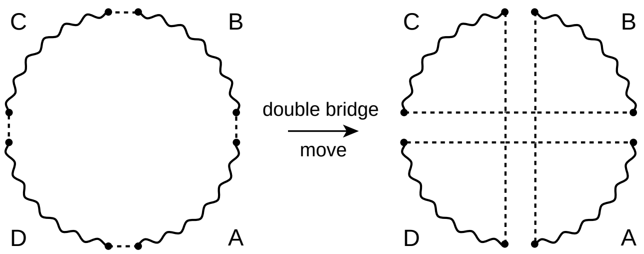
  ⇒ ILS performs random walk in the space of local optima reached by subsidiary local search.

- Intermediate behavior: select between the two candidate solutions based on the *Metropolis criterion* (*e.g.*, used in *Large Step Markov Chains* [Martin *et al.*, 1991].

- Advanced acceptance criteria take into account search history, *e.g.*, by occasionally reverting to *incumbent solution*.

Example: Iterated Local Search for the TSP (1)

- **Given:** TSP instance G.
- **Search space:** Hamiltonian cycles in G.
- **Subsidiary local search:** Lin-Kernighan variable depth search algorithm
- **Perturbation mechanism:**
  'double-bridge move' = particular 4-exchange step:



- **Acceptance criterion:** Always return the best of the two given candidate round trips.

Example: Iterated Local Search for the TSP (2)

Note:

- Double-bridge move local cannot be directly reversed by a sequence of 2-exchange steps as performed by "usual" LK implementations.
- This perturbation is empirically shown to be effective independent of instance size.

Note:

- This ILS algorithm for the TSP is known as *Iterated Lin-Kernighan (ILK) Algorithm*.

- Although ILK is structurally rather simple, an efficient implementation was shown to achieve excellent performance [Johnson and McGeoch, 1997].

Iterated local search algorithms ...

- are typically rather easy to implement (given existing implementation of subsidiary simple LS algorithms);
- achieve state-of-the-art performance on many combinatorial problems, including the TSP.

There are many LS approaches that are closely related to ILS, including:

- Large Step Markov Chains [Martin *et al.*, 1991]
- Chained Local Search [Martin and Otto, 1996]
- Variants of Variable Neighbourhood Search (VNS) [Hansen and Mladenovič, 2002]

Tabu Search

Key idea: Use aspects of search history (memory) to escape from local minima.

- Associate *tabu attributes* with candidate solutions or solution components.
- Forbid steps to search positions recently visited by underlying iterative best improvement procedure based on tabu attributes.

Tabu Search (TS):

determine initial candidate solution s
While *termination criterion* is not satisfied:
| *determine set N' of non-tabu neighbors of s*
| *choose a best candidate solution s' in N'*
|
| *update tabu attributes* based on s'
| $s := s'$

Example: Tabu Search for GCP – TabuCol

- **Search space:** set of all complete colorings of G.
- **Solution set:** proper colorings of G.
- **Neighborhood relation:** one-exchange.
- **Memory:** Associate tabu status (Boolean value) with each pair (v, c).
- **Initialization:** a construction heuristic
- **Search steps:**
  - pairs (v, c) are tabu if they have been changed in the last *tt* steps;
  - neighboring colorings are admissible if they can be reached by changing a non-tabu pair or have fewer unsatisfied edge constr. than the best coloring seen so far (*aspiration criterion*);
  - choose uniformly at random admissible coloring with minimal number of unsatisfied constraints.
- **Termination:** upon finding a proper coloring for G *or* after given bound on number of search steps has been reached *or* after a number of idle iterations

Note:

- Non-tabu search positions in $N(s)$ are called *admissible neighbors of s*.
- After a search step, the current search position or the solution components just added/removed from it are declared *tabu* for a fixed number of subsequent search steps (*tabu tenure*).
- Often, an additional *aspiration criterion* is used: this specifies conditions under which tabu status may be overridden (*e.g.*, if considered step leads to improvement in incumbent solution).
- Crucial for efficient implementation:
  - keep time complexity of search steps minimal by using special data structures, incremental updating and caching mechanism for evaluation function values;
  - efficient determination of tabu status: store for each variable x the number of the search step when its value was last changed it_x; x is tabu if $it - it_x < tt$, where it = current search step number.

Note: Performance of Tabu Search depends crucially on setting of tabu tenure *tt*:

- *tt* too low $\Rightarrow$ search stagnates due to inability to escape from local minima;
- *tt* too high $\Rightarrow$ search becomes ineffective due to overly restricted search path (admissible neighborhoods too small)

Advanced TS methods:

- **Robust Tabu Search** [Taillard, 1991]:
  repeatedly choose *tt* from given interval;
  also: force specific steps that have not been made for a long time.
- **Reactive Tabu Search** [Battiti and Tecchiolli, 1994]:
  dynamically adjust *tt* during search;
  also: use escape mechanism to overcome stagnation.

Further improvements can be achieved by using *intermediate-term* or *long-term memory* to achieve additional *intensification* or *diversification*.

Examples:

- Occasionally backtrack to *elite candidate solutions*, *i.e.*, high-quality search positions encountered earlier in the search; when doing this, all associated tabu attributes are cleared.
- Freeze certain solution components and keep them fixed for long periods of the search.
- Occasionally force rarely used solution components to be introduced into current candidate solution.
- Extend evaluation function to capture frequency of use of candidate solutions or solution components.

Tabu search algorithms algorithms are state of the art for solving many combinatorial problems, including:

- SAT and MAX-SAT
- the Constraint Satisfaction Problem (CSP)
- many scheduling problems

Crucial factors in many applications:

- choice of neighborhood relation
- efficient evaluation of candidate solutions
  (caching and incremental updating mechanisms)

Example: Tabu Search for QAP

- **Solution representation**: permutation π
- **Initial Solution**: randomly generated
- **Neighborhood**: interchange
  $\Delta_I : \quad \delta(\pi) = \{\pi' | \pi'_k = \pi_k \text{ for all } k \neq \{i,j\} \text{ and } \pi'_i = \pi_j, \pi'_j = \pi_i\}$
- **Tabu status**: forbid δ that place back the items in the positions they have already occupied in the last *tt* iterations (short term memory)
- Implementation details:
  - compute $g(\pi') - f(\pi)$ in $O(n)$ or $O(1)$ by storing the values all possible previous moves.
  - maintain a matrix $[T_{ij}]$ of size $n \times n$ and write the last time item i was moved in location k plus tt
  - δ is tabu if it satisfies both:
    - $T_{i,\pi(j)} \geq$ current iteration
    - $T_{j,\pi(i)} \geq$ current iteration

Example: Robust Tabu Search for QAP

- **Aspiration criteria**:
  - allow forbidden δ if it improves the last π^*
  - select δ if never chosen in the last A iterations (long term memory)
- Parameters: $tt \in [\lfloor 0.9n \rfloor, \lceil 1.1n + 4 \rceil]$ and $A = 5n^2$

Example: Reactive Tabu Search for QAP

- **Aspiration criteria**:
  - allow forbidden δ if it improves the last π^*
- **Tabu Tenure**
  - maintain a hash table (or function) to record previously visited solutions
  - increase *tt* by a factor $\alpha_{inc}(= 1.1)$ if the current solution was previously visited
  - decrease *tt* by a factor $\alpha_{dec}(= 0.9)$ if *tt* not changed in the last $sttc$ iterations or all moves are tabu
- Trigger escape mechanism if a solution is visited more than $nr(= 3)$ times
- Escape mechanism $= 1 + (1 + r) \cdot ma/2$ random moves