DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 4

LS Components. Beam Search, GRASP, Variable Neighborhood Search.

Marco Chiarandini

Outline

Solution Representations and Neighborhoods

Three different types of solution representations:

- *Permutation*
  - *linear permutation*: Single Machine Total Weighted Tardiness Problem
  - *circular permutation*: Traveling Salesman Problem
- *Assignment*: Graph Coloring Problem, SAT, CSP

A neighborhood function $N : S \to S \times S$ is also defined through an operator.
An operator Δ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(S) \quad \iff \quad \exists \delta \in \Delta | \delta(s) = s'$$

Permutations

$\Pi(n)$ indicates the set all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2 \ldots, n)$ is the identity permutation ι.

If $\pi \in \Pi(n)$ and $1 \leq i \leq n$ then:

- π_i is the element at position i
- $pos_\pi(i)$ is the position of element i

Alternatively, a permutation is a bijective function $\pi(i) = \pi_i$

the permutation product $\pi \cdot \pi'$ is the composition $(\pi \cdot \pi')_i = \pi'(\pi(i))$

For each π there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$

$$\Delta_N \subset \Pi$$

Neighborhood Operators for Linear Permutations

Swap operator

$$\Delta_S = \{\delta_S^i | 1 \leq i \leq n\}$$

$$\delta_S^i(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i+1} \pi_i \ldots \pi_n)$$

Interchange operator

$$\Delta_X = \{\delta_X^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_X^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{i+1} \ldots \pi_{j-1} \pi_i \pi_{j+1} \ldots \pi_n)$$

Insert operator

$$\Delta_I = \{\delta_I^{ij} | 1 \leq i \leq n, 1 \leq j \leq n, j \neq i\}$$

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_n) & i > j \end{cases}$$

Neighborhood Operators for Circular Permutations

Reversal (2-edge-exchange)

$$\Delta_R = \{\delta_R^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_R^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \ldots \pi_i \pi_{j+1} \ldots \pi_n)$$

Block moves (3-edge-exchange)

$$\Delta_B = \{\delta_B^{ijk} | 1 \leq i < j < k \leq n\}$$

$$\delta_B^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \ldots \pi_k \pi_i \ldots \pi_{j-1} \pi_{k+1} \ldots \pi_n)$$

Short block move (Or-edge-exchange)

$$\Delta_{SB} = \{\delta_{SB}^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_{SB}^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_i \ldots \pi_{j-1} \pi_{j+3} \ldots \pi_n)$$

Neighborhood Operators for Assignments

An assignment can be represented as a mapping
$\sigma : \{X_1 \ldots X_n\} \to \{v : v \in D, |D| = k\}$:

$$\sigma = \{X_i = v_i, X_j = v_j, \ldots\}$$

One exchange operator

$$\Delta_{1E} = \{\delta_{1E}^{il} | 1 \leq i \leq n, 1 \leq l \leq k\}$$

$$\delta_{1E}^{il}(\sigma) = \{\sigma : \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j) \ \forall j \neq i\}$$

Two exchange operator

$$\Delta_{2E} = \{\delta_{2E}^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_{2E}^{ij}\{\sigma : \sigma'(X_i) = \sigma(X_j), \ \sigma'(X_j) = \sigma(X_i) \text{ and } \sigma'(X_l) = \sigma(X_l) \ \forall l \neq i, j\}$$

Concepts that Define the Search Landscape

- Neighborhood structure $N : S \to S \times S$
  Often defined through an operator
- Neighborhood Graph $G_N = (S, E_N)$
- Distance d_N: length of shortest path between two solutions s and s' in G_N
- Evaluation Function $g(\pi) : S \mapsto \Re$

Definition:
The **Search Landscape** $\mathcal{L}(\pi)$ of an instance π is the triple

$$\mathcal{L}(\pi) = \langle S(\pi), N(\pi), g(\pi) \rangle$$

Distances

Set of paths in G_N with $s, s' \in S$:

$$\Phi(s, s') = \{(s_1, \ldots, s_h) | s_1 = s, s_h = s' \, \forall i : 1 \leq i \leq h-1, \langle s_i, s_{i+1} \rangle \in E_N\}$$

If $\phi = (s_1, \ldots, s_h) \in \Phi(s, s')$ let $|\phi| = h$ be the length of the path; then the **distance** between any two solutions s, s' is the length of shortest path between s and s' in G_N:

$$d_N(s, s') = \min_{\phi \in \Phi(s,s')} |\Phi|$$

Note: with permutations it is easy to see that:

$$d_N(\pi, \pi') = d_N(\pi^{-1} \cdot \pi', \iota)$$

Distances for Linear Permutation Representations

Swap neighborhood operator
computable in $O(n^2)$ by the precedence distance metric:
$\#\{\langle i, j \rangle | 1 \leq i < j \leq n, pos_{\pi'}(\pi_j) < pos_{\pi'}(\pi_i)\}$.
$diam(G_N) = n(n-1)/2$

Interchange neighborhood operator
Computable in $O(n) + O(n)$ since
$d_X(\pi, \pi') = d_X(\pi^{-1} \cdot \pi', \iota) = n - c(\pi^{-1} \cdot \pi')$ where $c(\pi)$ is the number of disjoint cycles that decompose a permutation.
$diam(G_{N_X}) = n - 1$

Insert neighborhood operator
Computable in $O(n) + O(n\, log(n))$ since
$d_I(\pi, \pi') = d_I(\pi^{-1} \cdot \pi', \iota) = n - |lis(\pi^{-1} \cdot \pi')|$ where $lis(\pi)$ denotes the length of the longest increasing subsequence.
$diam(G_{N_I}) = n - 1$

Distances for Circular Permutation Representations

Reversal neighborhood operator
sorting by reversal is known to be NP-hard

Block moves neighborhood operator
unknown whether it is NP-hard but there does not exist a proved polynomial-time algorithm

Distances for Assignment Representations

- Hamming Distance
- An assignment can be seen as a partition of n in k mutually exclusive non-empty subsets

  One-exchange neighborhood operator
  The *partition-distance* $d_{1E}(\mathcal{P}, \mathcal{P}')$ between two partitions $\mathcal{P}$ and $\mathcal{P}'$ is the minimum number of elements that must be moved between subsets in $\mathcal{P}$ so that the resulting partition equals $\mathcal{P}'$.

  The partition-distance can be computed in polynomial time by solving an assignment problem. Given the assignment matrix M where in each cell (i, j) it is $|S_i \cap S'_j|$ with $S_i \in \mathcal{P}$ and $S'_j \in \mathcal{P}'$ and defined $A(\mathcal{P}, \mathcal{P}')$ the assignment of maximal sum then it is $d_{1E}(\mathcal{P}, \mathcal{P}') = n - A(\mathcal{P}, \mathcal{P}')$

Simple Mechanisms for Escaping from Local Optima

- *Restart:* re-initialize search whenever a local optimum is encountered.
  (Often rather ineffective due to cost of initialization.)
- *Non-improving steps*: in local optima, allow selection of candidate solutions with equal or worse evaluation function value, *e.g.*, using minimally worsening steps.
  (Can lead to long walks in *plateaus*, *i.e.*, regions of search positions with identical evaluation function.)

Note: Neither of these mechanisms is guaranteed to always escape effectively from local optima.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- Intensification: aims to greedily increase solution quality or probability, *e.g.*, by exploiting the evaluation function.
- Diversification: aim to prevent search stagnation by preventing search process from getting trapped in confined regions.

Examples:

- Iterative Improvement (II): *intensification* strategy.
- Uninformed Random Walk/Picking (URW/P): *diversification* strategy.

Balanced combination of intensification and diversification mechanisms forms the basis for advanced LS methods.

Beam Search

Possible extension of tree based construction heuristics:
beam search/pilot method

- maintains a set B of bw (beam width) partial candidate solutions
- at each iteration extend extend each solution from B in fw (filter width) possible ways
- rank each $bw \times fw$ candidate solutions and take the best bw partial solutions
- complete candidate solutions obtained by B are maintained in B_f
- Stop when no partial solution in B is to be extend

Greedy Randomized Adaptive Search Procedure (GRASP)

Key Idea: Combine randomized constructive search with subsequent perturbative search.

Motivation:

- Candidate solutions obtained from construction heuristics can often be substantially improved by perturbative search.
- Perturbative search methods typically often require substantially fewer steps to reach high-quality solutions
  when initialized using greedy constructive search rather than random picking.
- By iterating cycles of constructive + perturbative search, further performance improvements can be achieved.

Greedy Randomized "Adaptive" Search Procedure (GRASP):

While *termination criterion* is not satisfied:
- generate candidate solution s using subsidiary greedy randomized constructive search
- perform subsidiary perturbative search on s

Note:

- Randomization in *constructive search* ensures that a large number of good starting points for *subsidiary perturbative search* is obtained.
- Constructive search in GRASP is 'adaptive' (or dynamic): Heuristic value of solution component to be added to given partial candidate solution r may depend on solution components present in r.
- Variants of GRASP without perturbative search phase (aka *semi-greedy heuristics*) typically do not reach the performance of GRASP with perturbative search.

Restricted candidate lists (RCLs)

- Each step of *constructive search* adds a solution component selected uniformly at random from a restricted candidate list (RCL).
- RCLs are constructed in each step using a *heuristic function* h.
- RCLs based on cardinality restriction comprise the k best-ranked solution components. (k is a parameter of the algorithm.)
- RCLs based on value restriction comprise all solution components l for which $h(l) \leq h_{min} + \alpha \cdot (h_{max} - h_{min})$, where h_{min} = minimal value of h and h_{max} = maximal value of h for any l. (α is a parameter of the algorithm.)

Example: GRASP for SAT [Resende and Feo, 1996]

- **Given:** CNF formula F over variables $x_1, \ldots, x_n$
- **Subsidiary constructive search:**
  - start from empty variable assignment
  - in each step, add one atomic assignment (*i.e.*, assignment of a truth value to a currently unassigned variable)
  - heuristic function $h(i, v) :=$ number of clauses that become satisfied as a consequence of assigning $x_i := v$
  - RCLs based on cardinality restriction (contain fixed number k of atomic assignments with largest heuristic values)
- **Subsidiary perturbative search:**
  - iterative best improvement using 1-flip neighborhood
  - terminates when model has been found or given number of steps has been exceeded

GRASP has been applied to many combinatorial problems, including:

- SAT, MAX-SAT
- the Quadratic Assignment Problem
- various scheduling problems

Extensions and improvements of GRASP:

- reactive GRASP (*e.g.*, dynamic adaptation of α during search)
- combinations of GRASP with Tabu Search and other LS methods

Variable Neighborhood Search (VNS)

Variable Neighborhood Search is an SLS method that is based on the systematic change of the neighborhood during the search.

Central observations

- a local minimum w.r.t. one neighborhood structure is not necessarily locally minimal w.r.t. another neighborhood structure
- a global optimum is locally optimal w.r.t. all neighborhood structures

- Principle: change the neighborhood during the search
- Several adaptations of this central principle
  - Variable Neighborhood Descent (VND)
  - Variable Neighborhood Search (VNS)
  - Reduced Variable Neighborhood Search (RVNS)
  - Variable Neighborhood Decomposition Search (VNDS)
  - Skewed Variable Neighborhood Search (SVNS)
- Notation
  - N_k, $k = 1, 2, \ldots, k_{max}$ is a set of neighborhood structures
  - $N_k(s)$ is the set of solutions in the k-th neighborhood of s

How to generate the various neighborhood structures?

- for many problems different neighborhood structures (local searches) exist / are in use
- change parameters of existing local search algorithms
- use k-exchange neighborhoods; these can be naturally extended
- many neighborhood structures are associated with distance measures; in this case increase the distance

Basic Variable Neighborhood Descent (BVND)

Procedure VND
input : N_k, $k = 1, 2, \ldots, k_{max}$, and an initial solution s
output: a local optimum s for N_k, $k = 1, 2, \ldots, k_{max}$
$k \leftarrow 1$
repeat
  $s' \leftarrow$ FindBestNeighbor(s,N_k)
  if $g(s') < g(s)$ **then**
    $s \leftarrow s'$
    $(k \leftarrow 1)$
  else
    $k \leftarrow k + 1$
until $k = k_{max}$;

Variable Neighborhood Descent (VND)

Procedure VND
input : N_k, $k = 1, 2, \ldots, k_{max}$, and an initial solution s
output: a local optimum s for N_k, $k = 1, 2, \ldots, k_{max}$
$k \leftarrow 1$
repeat
    $s' \leftarrow$ IterativeImprovement(s,N_k)
    if $g(s') < g(s)$ **then**
        $s \leftarrow s'$
        $(k \leftarrow 1)$
    else
        $k \leftarrow k + 1$
until $k = k_{max}$;

- Final solution is locally optimal w.r.t. all neighborhoods
- First improvement may be applied instead of best improvement
- Typically, order neighborhoods from smallest to largest
- If iterative improvement algorithms II_k, $k = 1, \ldots, k_{max}$ are available as black-box procedures
  - order black-boxes
  - apply them in the given order
  - possibly iterate starting from the first one
  - order chosen by: *solution quality* and *speed*

Example

VND for single-machine total weighted tardiness problem

- Candidate solutions are permutations of job indexes
- Two neighborhoods: swap and insert
- Influence of different starting heuristics also considered

initial	swap		insert		swap+insert		insert+swap	
solution	Δavg	$t$avg	Δavg	$t$avg	Δavg	$t$avg	Δavg	$t$avg
EDD	0.62	0.140	1.19	0.64	0.24	0.20	0.47	0.67
MDD	0.65	0.078	1.31	0.77	0.40	0.14	0.44	0.79

Δavg deviation from best-known solutions, averaged over 100 instances

Basic Variable Neighborhood Search ((B)VNS)

Procedure VND
input : N_k, $k = 1, 2, \ldots, k_{max}$, and an initial solution s
output: a local optimum s for N_k, $k = 1, 2, \ldots, k_{max}$
repeat
    $k \leftarrow 1$
    repeat
        $s' \leftarrow$ RandomPicking(s,N_k)
        $s'' \leftarrow$ IterativeImprovement(s',N_1)
        if $g(s'') < g(s)$ **then**
            $s \leftarrow s''$
            $k \leftarrow 1$
        else
            $k \leftarrow k + 1$
    until $k = k_{max}$;
until Termination Condition ;

To decide:

- which neighborhoods
- how many
- which order
- which change strategy

- Extended version: parameters k_{min} and k_{step}; set $k \leftarrow k_{min}$ and increase by k_{step} if no better solution is found

Extensions (1)

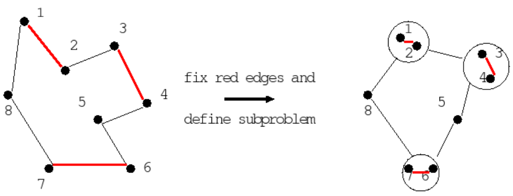
Reduced Variable Neighborhood Search (RVNS)

- same as VNS except that no IterativeImprovement procedure is applied
- only explores randomly different neighborhoods
- can be faster than standard local search algorithms for reaching good quality solutions

Extensions (2)

Variable Neighborhood Decomposition Search (VNDS)

- same as in VNS but in IterativeImprovement all solution components are kept fixed except k randomly chosen
- IterativeImprovement is applied on the k unfixed components



- IterativeImprovement can be substituted by exhaustive search up to a maximum size b (parameter) of the problem

Extensions (3)

Skewed Variable Neighborhood Search (SVNS)

- Set $s \leftarrow s''$ even if s'' it is worse
  - according to some probability
  - Skewed VNS: accept if

$$g(s) - \alpha \cdot d(s, s'') < g(s)$$

$d(s, s'')$ measure the distance between solutions
(underlying idea: avoiding degeneration to multi-start)

Outline

3. Example Problems
   The Single Machine Total Tardiness Problem

Example Problems: So far...

- Traveling Salesman Problem (TSP)
- Vertex Coloring Problem (GCP)
- Propositional Satisfiability (SAT and MAX-SAT)
- Constraint Satisfaction Problem (CSP and MAX-CSP)

The Single Machine Total Tardiness Problem

Given: a set of n jobs $\{J_1, \ldots, J_n\}$ to be processed on a single machine and for each job J_i a processing time p_i, a weight w_i and a due date d_i.

Task: Find a schedule that minimizes the total weighted tardiness $\sum_{i=1}^{n} w_i \cdot T_i$ where $T_i = \{C_i - d_i, 0\}$ (C_i completion time of job J_i)

Example:

Job	J_1	J_2	J_3	J_4	J_5	J_6
Processing Time	3	2	2	3	4	3
Due date	6	13	4	9	7	17
Weight	2	3	1	5	1	2

Sequence $\phi = J_3, J_1, J_5, J_4, J_1, J_6$

Job	J_3	J_1	J_5	J_4	J_1	J_6
C_i	2	5	9	12	14	17
T_i	0	0	2	3	1	0
$w_i \cdot T_i$	0	0	2	15	3	0