Lecture 2

Combinatorial Problems and Local Search

Marco Chiarandini
Outline

1. Example Problems
   The Graph Colouring
   The Satisfiability Problem

2. Local Search Methods
Example Problems

- They are chosen because conceptually concise, intended to illustrate the development, analysis and presentation of algorithms.

- Although real-world problems tend to have much more complex formulations these problems capture their essence.
The Vertex Coloring Problem

Given: A graph $G$ and a set of colors $\Gamma$.

A *proper coloring* is an assignment of one color to each vertex of the graph such that adjacent vertices receive different colors.

### Decision version ($k$-coloring)

Task: Find a proper coloring of $G$ which uses at most $k$ colors.

### Optimization version (chromatic number)

Task: Find a proper coloring of $G$ which uses the minimal number of colors.
The SAT Problem

General SAT Problem (search variant):

- **Given:** Formula $F$ in propositional logic
- **Objective:** Find an assignment of truth values to variables in $F$ that renders $F$ true, or decide that no such assignment exists.

SAT: A simple example

- **Given:** Formula $F := (x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2)$
- **Objective:** Find an assignment of truth values to variables $x_1, x_2$ that renders $F$ true, or decide that no such assignment exists.

MAX-SAT

Which is the maximal number of clauses satisfiable in a propositional logic formula $F$?
Definition:

- **Formula in propositional logic**: well-formed string that may contain
  - propositional variables $x_1, x_2, \ldots, x_n$;
  - truth values $\top$ ('true'), $\bot$ ('false');
  - operators $\neg$ ('not'), $\land$ ('and'), $\lor$ ('or');
  - parentheses (for operator nesting).

- **Model** (or **satisfying assignment**) of a formula $F$: Assignment of truth values to the variables in $F$ under which $F$ becomes true (under the usual interpretation of the logical operators)

- Formula $F$ is **satisfiable** iff there exists at least one model of $F$, **unsatisfiable** otherwise.
Definition:

- A formula is in **conjunctive normal form (CNF)** iff it is of the form

\[
\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k(i)} l_{ij} = (l_{11} \lor \ldots \lor l_{1k(1)}) \ldots \land (l_{m1} \lor \ldots \lor l_{mk(m)})
\]

where each literal \(l_{ij}\) is a propositional variable or its negation. The disjunctions \((l_{i1} \lor \ldots \lor l_{ik(i)})\) are called **clauses**.

- A formula is in **\(k\)-CNF** iff it is in CNF and all clauses contain exactly \(k\) literals (i.e., for all \(i\), \(k(i) = k\)).

- In many cases, the restriction of SAT to CNF formulae is considered.

- The restriction of SAT to \(k\)-CNF formulae is called **\(k\)-SAT**.

- For every propositional formula, there is an equivalent formula in 3-CNF.
Example:

\[ F := \land (\neg x_2 \lor x_1) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2) \land (\neg x_4 \lor x_3) \land (\neg x_5 \lor x_3) \]

- \( F \) is in CNF.
- Is \( F \) satisfiable?
  Yes, e.g., \( x_1 := x_2 := \top, x_3 := x_4 := x_5 := \bot \) is a model of \( F \).
Many combinatorial problems are hard but some problems can be solved efficiently

- Longest path problem is $\mathcal{NP}$-hard but not shortest path problem

- SAT for 3-CNF is $\mathcal{NP}$-complete but not 2-CNF (linear time algorithm)

- TSP is $\mathcal{NP}$-hard, the associated decision problem (for any solution quality) is $\mathcal{NP}$-complete but not the Euler tour problem

- TSP on Euclidean instances is $\mathcal{NP}$-hard but not where all vertices lie on a circle.

- The Graph Coloring Problem is $\mathcal{NP}$-complete but not on interval graphs

- Many scheduling and timetabling problems are $\mathcal{NP}$-hard
Incomplete Search Paradigms

Construction Heuristics
An important class of Construction Heuristics are greedy algorithms.

- **Strategy:** always make the choice which is the best at the moment.
- **They are not generally guaranteed to find globally optimal solutions (but sometimes they do: Minimum Spanning Tree, Single Source Shortest Path, etc.)**

Possible extension of construction heuristics: **beam search/pilot method**
- maintains a set $B$ of $bw$ (beam width) partial candidate solutions
- at each level extend $fw$ (filter width) candidate solutions and rank
- complete candidate solutions obtained by $B$ are maintained in $B_f$
Incomplete Search Paradigms

Perturbative search

- search space = complete candidate solutions
- search step = modification of one or more solution components
- iteratively generate and evaluate candidate solutions
  - decision problems: evaluation = test if solution
  - optimization problems: evaluation = check objective function value
- evaluating candidate solutions is typically computationally much cheaper than finding (optimal) solutions

Iterative Improvement (II):

While $s$ has better neighbors:
  
  choose a neighbor $s'$ of $s$
  such that $g(s') < g(s)$
  $s := s'$
Local Search Methods

In order to obtain very high quality solutions it is often successful to combine the two heuristic paradigms.

Meta-heuristics
General guidance criteria over lower level heuristics.

- Metaheuristics
  - Construction Heuristics
  - Perturbative Search

Informed search based on local or incomplete knowledge as opposed to systematic search ⇒ Local Search Methods

Stochastic Local Search (SLS) algorithms use randomized choices in generating and modifying candidate solutions. They are introduced whenever it is unknown which deterministic rules are profitable for all the instances of interest.
Example: Uninformed random walk for $k$-coloring

**procedure** \textit{URW-for-$k$-col}($G, k, \text{maxSteps}$)

\begin{algorithmic}
\STATE \textbf{input:} graph \textit{G}, integer \textit{k}, integer \textit{maxSteps}
\STATE \textbf{output:} feasible coloring of \textit{G} or $\emptyset$
\STATE choose assignment $\varphi$ of \textit{k} colors to all vertices in \textit{G} uniformly at random;
\STATE \textit{steps} := 0;
\WHILE{not ([\varphi is a proper coloring] \textbf{and} (\textit{steps} < \textit{maxSteps}))}
\STATE randomly select vertex \textit{v} in \textit{G};
\STATE change value of $\varphi(\textit{v})$;
\STATE \textit{steps} := \textit{steps}+1;
\ENDWHILE
\IF{$\varphi$ is feasible}
\STATE \textbf{return} $\varphi$
\ELSE
\STATE \textbf{return} $\emptyset$
\ENDIF
\end{algorithmic}

end \textit{URW-for-$k$-col}
**Systematic search** is often better suited when ...  
- proofs of insolubility or optimality are required;  
- time constraints are not critical;  
- problem-specific knowledge can be exploited.

**Local search** is often better suited when ...  
- non linear constraints and non linear objective function;  
- reasonably good solutions are required within a short time;  
- problem-specific knowledge is rather limited.

**Complementarity:**  
Local and systematic search can be fruitfully combined, *e.g.*, by using local search for finding solutions whose optimality is proved using systematic search.
Local search — global view

- vertices: candidate solutions (search positions)
- edges: connect neighboring positions
- s: (optimal) solution
- c: current search position
Definition: Local Search Algorithm (1)

For given problem instance $\pi$:

- **search space** $S(\pi)$
  (e.g., for GCP: assignment of colors to each vertex)

- **solution set** $S'(\pi) \subseteq S(\pi)$
  (e.g., for GCP: feasible assignment)

- **neighborhood relation** $N(\pi) \subseteq S(\pi) \times S(\pi)$
  (e.g., for GCP: neighboring assignments differ in the color at one vertex)
Definition: Local Search Algorithm (2)

- **set of memory states** $M(\pi)$
  (may consist of a single state, for LS algorithms that do not use memory)

- **initialization function** $\text{init} : \emptyset \mapsto \mathcal{D}(S(\pi) \times M(\pi))$
  (specifies probability distribution over initial search positions and memory states)

- **step function** $\text{step} : S(\pi) \times M(\pi) \mapsto \mathcal{D}(S(\pi) \times M(\pi))$
  (maps each search position and memory state onto probability distribution over subsequent, neighboring search positions and memory states)

- **termination predicate** $\text{terminate} : S(\pi) \times M(\pi) \mapsto \mathcal{D}(\{\top, \bot\})$
  (determines the termination probability for each search position and memory state)
procedure \textit{LS-Decision}(\pi) \\
\textbf{input:} problem instance \(\pi \in \Pi\) \textbf{output:} solution \(s \in S'(\pi)\) or \(\emptyset\) \\
\( (s, m) := \text{init}(\pi); \)

\textbf{while not} \textit{terminate}(\pi, s, m) \textbf{do} \\\n\hspace{1em} (s, m) := \text{step}(\pi, s, m); \textbf{end} \\

\textbf{if} \(s \in S'(\pi)\) \textbf{then} \\\n\hspace{1em} \text{return } s \\
\textbf{else} \hspace{1em} \text{return } \emptyset \textbf{end} \textbf{end} \textit{LS-Decision}
procedure $LS$-Minimization($\pi'$)
\begin{itemize}
\item[] input: problem instance $\pi' \in \Pi'$
\item[] output: solution $s \in S'(\pi')$ or $\emptyset$
\item[]
\end{itemize}

$(s, m) := init(\pi')$;

$\hat{s} := s$;

while not $terminate(\pi', s, m)$ do

$(s, m) := step(\pi', s, m)$;

if $f(\pi', s) < f(\pi', \hat{s})$ then

$\hat{s} := s$;

end

end

if $\hat{s} \in S'(\pi')$ then

return $\hat{s}$

else

return $\emptyset$

end

end $LS$-Minimization