

Foundations of Artificial Intelligence

20. Combinatorial Optimization: Introduction and Hill-Climbing

Malte Helmert

Universität Basel

April 8, 2016

Combinatorial Optimization

Introduction

previous chapters: classical state-space search

- find action sequence (path) from initial to goal state
- difficulty: large number of states (“state explosion”)

next chapters: combinatorial optimization

↪ similar scenario, but:

- no actions or transitions
- don't search for path, but for configuration (“state”) with low cost/high quality

German: Zustandsraumexplosion, kombinatorische Optimierung, Konfiguration

Combinatorial Optimization: Overview

Chapter overview: combinatorial optimization

- 20. Introduction and Hill-Climbing
- 21. Advanced Techniques

Combinatorial Optimization Problems

Definition (combinatorial optimization problem)

A **combinatorial optimization problem** (COP) is given by a tuple $\langle C, S, opt, v \rangle$ consisting of:

- a set of (solution) **candidates** C
- a set of **solutions** $S \subseteq C$
- an **objective sense** $opt \in \{\min, \max\}$
- an **objective function** $v : S \rightarrow \mathbb{R}$

German: kombinatorisches Optimierungsproblem, Kandidaten, Lösungen, Optimierungsrichtung, Zielfunktion

Remarks:

- “problem” here in another sense (= “instance”) than commonly used in computer science
- practically interesting COPs usually have too many candidates to enumerate explicitly

Optimal Solutions

Definition (optimal)

Let $\mathcal{O} = \langle C, S, opt, v \rangle$ be a COP.

The **optimal solution quality** v^* of $\mathcal{O}$ is defined as

$$v^* = \begin{cases} \min_{c \in S} v(c) & \text{if } opt = \min \\ \max_{c \in S} v(c) & \text{if } opt = \max \end{cases}$$

(v^* is undefined if $S = \emptyset$.)

A solution s of $\mathcal{O}$ is called **optimal** if $v(s) = v^*$.

German: optimale Lösungsqualität, optimal

Combinatorial Optimization

The basic algorithmic problem we want to solve:

Combinatorial Optimization

Find a **solution** of good (ideally, optimal) quality for a combinatorial optimization problem $\mathcal{O}$ or prove that no solution exists.

Good here means **close to v^*** (the closer, the better).

Relevance and Hardness

- There is a huge number of practically important combinatorial optimization problems.
- Solving these is a central focus of **operations research**.
- Many important combinatorial optimization problems are **NP-complete**.
- Most “classical” NP-complete problems can be formulated as combinatorial optimization problems.

↪ **Examples:** TSP, VERTEXCOVER, CLIQUE, BINPACKING, PARTITION

German: Unternehmensforschung, NP-vollständig

Search vs. Optimization

Combinatorial optimization problems have

- a **search aspect** (among all candidates C , find a solution from the set S) and
- an **optimization aspect** (among all solutions in S , find one of high quality).

Pure Search/Optimization Problems

Important special cases arise when one of the two aspects is trivial:

- **pure search problems:**
 - all solutions are of equal quality
 - difficulty is in finding a solution **at all**
 - **formally:** v is a constant function (e.g., constant 0);
 opt can be chosen arbitrarily (does not matter)
- **pure optimization problems:**
 - all candidates are solutions
 - difficulty is in finding solutions of **high quality**
 - **formally:** $S = C$

Example

Example: 8 Queens Problem

8 Queens Problem

How can we

- place **8 queens** on a chess board
- such that **no two queens threaten each other?**

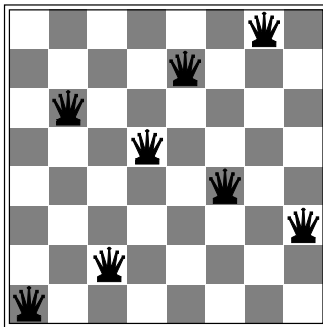
German: 8-Damen-Problem

- originally proposed in 1848
- **variants:** board size; other pieces; higher dimension

There are **92 solutions**, or **12 solutions** if we do not count symmetric solutions (under rotation or reflection) as distinct.

Example: 8 Queens Problem

Problem: Place 8 queens on a chess board such that no two queens threaten each other.



Is this candidate a solution?

Formally: 8 Queens Problem

How can we formalize the problem?

idea:

- obviously there must be exactly one queen in each file (“column”)
- describe candidates as 8-tuples, where the i -th entry denotes the rank (“row”) of the queen in the i -th file

formally: $\mathcal{O} = \langle C, S, opt, v \rangle$ with

- $C = \{1, \dots, 8\}^8$
- $S = \{ \langle r_1, \dots, r_8 \rangle \mid \forall 1 \leq i < j \leq 8 : r_i \neq r_j \wedge |r_i - r_j| \neq |i - j| \}$
- v constant, opt irrelevant (pure search problem)

Local Search: Hill Climbing

Algorithms for Combinatorial Optimization Problems

How can we algorithmically solve COPs?

- formulation as classical state-space search
- formulation as constraint network
- formulation as logical satisfiability problem
- formulation as mathematical optimization problem (LP/IP)
- local search

Algorithms for Combinatorial Optimization Problems

How can we algorithmically solve COPs?

- formulation as classical state-space search
 $\rightsquigarrow$ previous chapters
- formulation as constraint network $\rightsquigarrow$ next week
- formulation as logical satisfiability problem $\rightsquigarrow$ later
- formulation as mathematical optimization problem (LP/IP)
 $\rightsquigarrow$ not in this course
- local search $\rightsquigarrow$ this chapter and next chapter

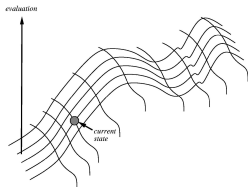
Search Methods for Combinatorial Optimization

- main ideas of **heuristic search** applicable for COPs
 $\rightsquigarrow$ states $\approx$ candidates
- main difference: no “actions” in problem definition
 - instead, **we** (as algorithm designers) can choose which candidates to consider **neighbors**
 - definition of neighborhood **critical aspect** of designing good algorithms for a given COP
- “path to goal” irrelevant to the user
 - no path costs, parents or generating actions
 - $\rightsquigarrow$ no search nodes needed

Local Search: Idea

main ideas of local search algorithms for COPs:

- heuristic h estimates quality of candidates
 - for pure optimization: often objective function v itself
 - for pure search: often distance estimate to closest solution (as in state-space search)
- do not remember paths, only candidates
- often only **one** current candidate $\rightsquigarrow$ very memory-efficient (however, not complete or optimal)
- often initialization with **random** candidate
- iterative improvement by **hill climbing**



Hill Climbing

Hill Climbing (for Maximization Problems)

current := a random candidate

repeat:

next := a neighbor of *current* with maximum *h* value

if $h(\textit{next}) \leq h(\textit{current})$:

return *current*

current := *next*

Remarks:

- search as **walk** “uphill” in a **landscape** defined by the **neighborhood relation**
- heuristic values define “height” of terrain
- analogous algorithm for minimization problems also traditionally called “hill climbing” even though the metaphor does not fully fit

Properties of Hill Climbing

- always terminates if candidate set is finite (Why?)
- no guarantee that result is a solution
- if result is a solution, it is **locally optimal** w.r.t. h , but no global quality guarantees

Example: 8 Queens Problem

Problem: Place 8 queens on a chess board
such that no two queens threaten each other.

possible heuristic: no. of pairs of queens threatening each other
(formalization as minimization problem)

possible neighborhood: move one queen within its file

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♚	13	16	13	16
♚	14	17	15	♚	14	16	16
17	♚	16	18	15	♚	15	♚
18	14	♚	15	15	14	♚	16
14	14	13	17	12	14	12	18

Performance of Hill Climbing for 8 Queens Problem

- problem has $8^8 \approx 17$ million candidates
(reminder: 92 solutions among these)
- after random initialization, hill climbing finds a solution
in around 14% of the cases
- only around 4 steps on average!

Summary

Summary

combinatorial optimization problems:

- find **solution** of good **quality** (objective value) among many **candidates**
- special cases:
 - pure search problems
 - pure optimization problems
- differences to state-space search:
no actions, paths etc.; only “state” matters

often solved via **local search**:

- consider **one candidate** (or a few) at a time;
try to improve it iteratively