Evolutionary Computing

Chapter 13

Chapter 13: Constraint Handling

- Motivation and the trouble
- What is a constrained problem?
- Evolutionary constraint handling
- A selection of related work
- Conclusions, observations, and suggestions

Motivation

Why bother about constraints?

- Practical relevance:
  a great deal of practical problems are constrained.
- Theoretical challenge:
  a great deal of untractable problems (NP-hard etc.) are constrained.

Why try with evolutionary algorithms?

- EAs show a good ratio of (implementation) effort/performance.
- EAs are acknowledged as good solvers for tough problems.

What is a constrained problem?

Consider the Travelling Salesman Problem for n cities, C = {$city_1$, … , $city_n$}

If we define the search space as

- S = C^n, then we need a constraint requiring uniqueness of each city in an element of S

- S = {permutations of C}, then we need no additional constraint.

The notion ‘constrained problem’ depends on what we take as search space

What is constrained search?
or
What is free search?

- Even in the space S = {permutations of C} we cannot perform free search

- **Free search**: standard mutation and crossover preserve membership of S, i.e., mut(x) $\in$ S and cross(x,y) $\in$ S

The notion 'free search' depends on what we take as standard mutation and crossover.

- mut is **standard mutation** if for all $\langle x_1, \ldots, x_n \rangle$, if mut($\langle x_1, \ldots, x_n \rangle$) = $\langle x'_1, \ldots, x'_n \rangle$, then $x'_i \in$ domain(i)

- cross is **standard crossover** if for all $\langle x_1, \ldots, x_n \rangle$, $\langle y_1, \ldots, y_n \rangle$, if cross($\langle x_1, \ldots, x_n \rangle$, $\langle y_1, \ldots, y_n \rangle$) = $\langle z_1, \ldots, z_n \rangle$, then
  - $z_i \in \{x_i, y_i\}$ discrete case
  - $z_i \in [x_i, y_i]$ continuous case

Free search space

Free search space: $S = D_1 \times \ldots \times D_n$

- one assumption on D_i: if it is continuous, it is convex
- the restriction $s_i \in D_i$ is not a constraint, it is the definition of the domain of the i-th variable
- membership of S is coordinate-wise, hence a free search space allows free search

A problem can be defined through

- an objective function (to be optimized)
- constraints (to be satisfied)

Types of problems

	Objective Function	
Constraints	Yes	No
Yes	Constrained optimization problem	Constraint satisfaction problem
No	Free optimization problem	No Problem

Free Optimization Problems

Free Optimization Problem: $\langle S, f, \bullet \rangle$

- S is a free search space
- f is a (real valued) objective function on S

Solution of an FOP: $s \in S$ such that f (s) is optimal in S

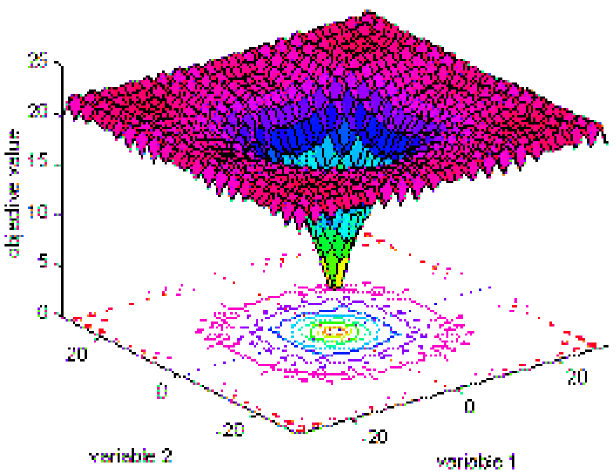
FOPs are ‘easy’, in the sense that:

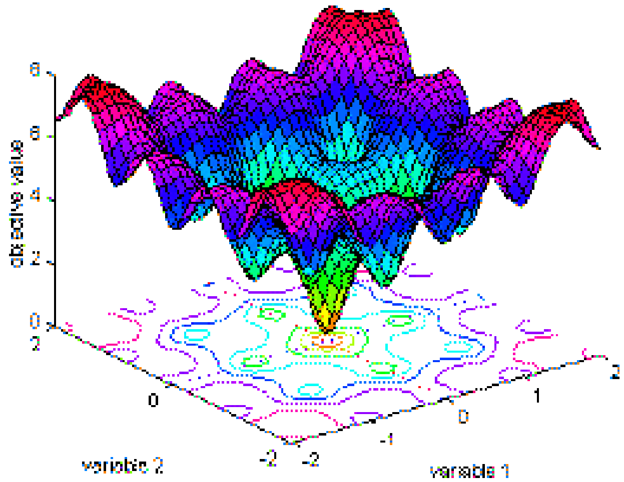
- it's ‘only’ optimizing, no constraints and
- EAs have a basic instinct for optimization

FOP: Example

Ackley function

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$





Constraint Satisfaction Problems (1/2)

Constraint Satisfaction Problem: $\langle S, \bullet, \Phi \rangle$

- S is a free search space
- Φ is a formula (Boolean function on S)

Φ is the **feasibility condition**

$S_{\Phi} = \{s \in S \mid \Phi(s) = \text{true}\}$ is the **feasible search space**

Solution of a CSP:

$s \in S$ such that $\Phi(s) = \text{true}$ (s is feasible)

Constraint Satisfaction Problems (2/2)

Φ is typically given by a set (conjunction) of constraints

- $c_i = c_i(x_{j1}, \ldots, x_{jni})$, where n_i is the arity of c_i
- $c_i \subseteq D_{j1} \times \ldots \times D_{jni}$ is also a common notation

FACTS:

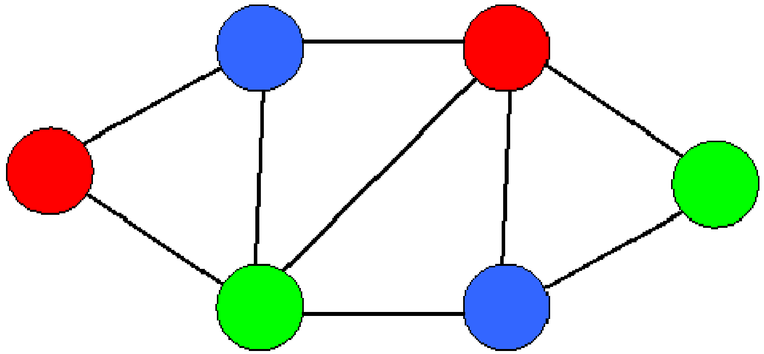
- The general CSP is NP-complete
- Every CSP is equivalent with a binary CSP, where all $n_i \equiv 2$
- Constraint density and constraint tightness are parameters that determine how hard an instance is

CSP: Example

Graph 3-coloring problem:

- $G = (N, E)$, $E \subseteq N \times N$, $|N| = n$
- $S = D^{\mathbf{n}}$, $D = \{1, 2, 3\}$
- $\Phi(s) = \Lambda_{\mathbf{e} \in \mathbf{E}}\, c_{\mathbf{e}}(s)$, where

  $c_{\mathbf{e}}(s) = \text{true}$ iff $e = (k, l)$ and $s_{\mathbf{k}} \neq s_{\mathbf{l}}$



Constrained optimization problems

Constrained Optimization Problem: $\langle S, f, \Phi \rangle$

- S is a free search space
- f is a (real valued) objective function on S
- Φ is a formula (Boolean function on S)

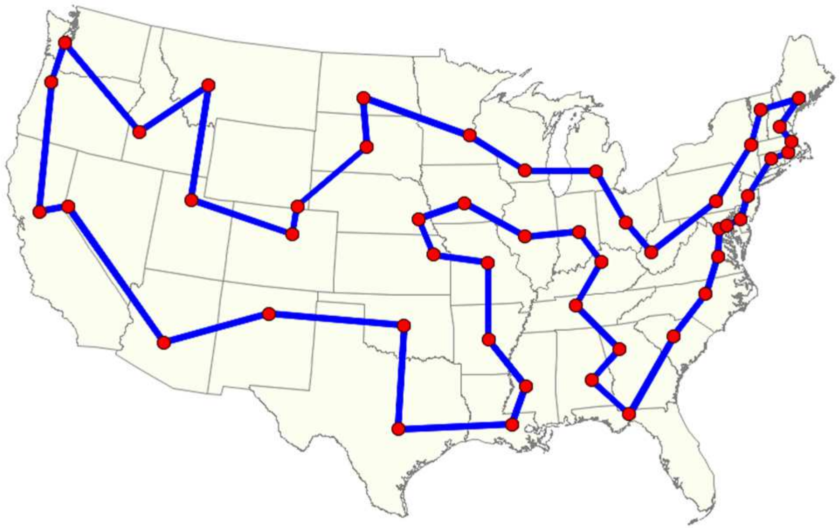
Solution of a COP:

$s \in S_{\Phi}$ such that f(s) is optimal in S_{Φ}

COP: Example

Travelling salesman problem

- $S = C^n$, $C = \{city_1, \ldots, city_n\}$
- $\Phi(s) = \text{true} \Leftrightarrow \forall i, j \in \{1, \ldots, n\}\ i \neq j \Rightarrow s_i \neq s_j$
- $f(s) = \sum_{i=1}^{n} dist(s_i, s_{i+1}), \quad \text{where } s_{n+1} := s_1$



Solving CSPs by EAs (1)

EAs need an f to optimize → $\langle S, \bullet, \Phi \rangle$ must be transformed first to a

1. FOP: $\langle S, \bullet, \Phi \rangle \rightarrow \langle S, f, \bullet \rangle$ or
2. COP: $\langle S, \bullet, \Phi \rangle \rightarrow \langle S, f, \Psi \rangle$

The transformation must be (semi-)equivalent, i.e. at least:

1. f (s) is optimal in S $\Rightarrow \Phi(s)$
2. ψ (s) and f (s) is optimal in S $\Rightarrow \Phi(s)$

Constraint handling

'Constraint handling' interpreted as 'constraint transformation'

Case 1: CSP → FOP

All constraints are handled **indirectly**, i.e., Φ is transformed into f and later they are solved by `simply' optimizing in ⟨S, f, •⟩

Case 2: CSP → COP

Some constraints handled **indirectly** (those transformed into f)

Some constraints handled **directly** (those remaining constraints in ψ)

In the latter case we also have 'constraint handling' in the sense of 'treated during the evolutionary search'

Indirect constraint handling: Introduction (1/3)

Constraint handling has two meanings:

1. how to **transform** the constraints in Φ into f, respectively ⟨f, ψ⟩ **before** applying an EA
2. how to **enforce** the constraints in ⟨S, f, Φ⟩ **while** running an EA

Case 1: constraint handling only in the 1st sense (pure penalty approach)

Case 2: constraint handling in both senses

In Case 2 the question

‘How to solve CSPs by EAs’

transforms to

‘How to solve COPs by EAs’

Indirect constraint handling: Introduction (2/3)

Note: **always** needed

- for all constraints in Case 1
- for some constraints in Case 2

Some general options

a. penalty for violated constraints

b. penalty for wrongly instantiated variables

c. estimating distance/cost to feasible solution

Indirect constraint handling: Introduction (3/3)

Notation:

- c_i constraints, i = {1, … , m}
- v_j variables, j = {1, … , n}
- C^j is the **set of constraints involving variable** v_j
- $S_c = \{z \in D_{j1} \times \ldots \times D_{jk} \mid c(z) = \text{true}\}$
  is the **projection** of c, if $v_{j1}, \ldots , v_{jk}$ are the var's of c
- $d(s, S_c) := \min\{d(s, z) \mid z \in 2\ S_c\}$
  is the **distance** of $s \in S$ from S_c (s is projected too)

Indirect constraint handling (3)

Formally:

a. $$f(\bar{s}) = \sum_{i=1}^{m} w_i \cdot \chi(\bar{s}, c_i), \text{ where}$$

$$\chi(\bar{s}, c_i) = \begin{cases} 1 & \text{if } \bar{s} \text{ violates } c_i \\ 0 & \text{otherwise} \end{cases}$$

b. $$f(\bar{s}) = \sum_{j=1}^{n} w_j \cdot \chi(\bar{s}, C^j), \text{ where}$$

$$\chi(\bar{s}, C^j) = \begin{cases} 1 & \text{if } \bar{s} \text{ violates at least one } c \in C^j \\ 0 & \text{otherwise} \end{cases}$$

c. $$f(\bar{s}) = \sum_{i=1}^{m} w_i \cdot d(\bar{s}, S_{c_i}), \text{ where}$$

Observe that for each option: $\forall \bar{s} \in S : \Phi(\bar{s}) \Leftrightarrow f(\bar{s}) = 0$

Indirect constraint handling: Graph coloring example

a. $f(\bar{s}) = \sum_{i=1}^{m} w_i \cdot \chi(\bar{s}, c_i)$, counts the 'wrong' edges

b. $f(\bar{s}) = \sum_{j=1}^{n} w_j \cdot \chi(\bar{s}, C^j)$, counts the 'wrong' nodes

c. $f(\bar{s}) = \sum_{i=1}^{m} w_i \cdot d(\bar{s}, S_{c_i})$, counts the 'wrong' edges

if we take the number of necessary corrections (recolorings) as distance.

Indirect constraint handling: pro’s & con’s

PRO’s of indirect constraint handling:

- conceptually simple, transparent
- problem independent
- reduces problem to ‘simple’ optimization
- allows user to tune on his/her preferences by weights
- allows EA to tune fitness function by modifying weights during the search

CON’s of indirect constraint handling:

- loss of info by packing everything in a single number
- said not to work well for sparse problems

Direct constraint handling (1/2)

Options:

- eliminating infeasible candidates (very inefficient, hardly practicable)
- repairing infeasible candidates
- preserving feasibility by special operators
  (requires feasible initial population
- decoding, i.e. transforming the search space
  (allows usual representation and operators)

Direct constraint handling (2/2)

PRO's of direct constraint handling:

- it works well (except eliminating)

CON's of direct constraint handling:

- problem specific
- no guidelines