Nature-Inspired Computing

Handling Constraints

Dr. Şima Uyar

September 2006

Introduction

- many practical problems are constrained
- not all combinations of variable values represent valid solutions
  - feasible solutions
  - infeasible solutions
- constraint handling not straightforward
  - most variation operators blind to constraints

Terminology

- problem given in terms of variables (v_1,....,v_n)
- each variable has domains (D_1,...,D_n)
- free search space: $S=D_1xD_2x...xD_n$
- problems distinguished by presence / absence of
  - an objective function
  - constraints

Problem Types

	Yes	No
Yes	Constrained Optimization Poblem	Constraint Satisfaction Problem
No	Free Optimization Problem	---

Free Optimization Problem - FOP

- defined by the pair <S,f>
  - S: free search space
  - f: objective function on S
- solution of an FOP is

$$\bar{s} \in S \quad with \quad optimal \quad f$$

Constraint Satisfaction Problems - CSP

- defined by the pair <S, Φ >
  - S: free search space
  - Φ : a formula (Boolean function on S)
    - usually called a feasibility condition
    - typically compound entity derived from more elementary constraints
- solution to a CSP:

$$\bar{s} \in S \quad with \quad \phi(\bar{s}) = True$$

CSP Examples

- graph three-coloring problem
  - G=(N,E), E⊆NxN
  - color the nodes of a graph G with 3 colors in such a way that no neighbour nodes have the same color
    - neighbour node: nodes connected by an edge
- boolean satisfiability problem (SAT)

CSP

- main challenges
  - no objective function to define fitness
  - extreme case of needle-in-a-haystack-problem
    - large plateaus at zero level (False)
    - some singular peaks (True)
- basic approach
  - transform constraints into optimization objectives

Constrained Optimization Problem - COP

- combination of FOP and CSP
- defined by a triple <S,f,ϕ>
  - S: free search space
  - f: objective function on S
  - ϕ: a formula (Boolean function on S)

COP Example - TSP

- n cities $C=(c_1,c_2,...,c_n)$
- $S=C^n$
- objective function: minimization

$$f(\bar{s}) = \sum_{i=1}^{n} d(s_i, s_{i+1}) \quad with \quad s_{n+1} \quad defined \quad as \quad s_1$$

COP

- similar to CSP
  - transform constraints into optimization function
    - becomes FOP
  - indirect constraint handling
    - done before NIH run
- leave constraints as constraints
  - explicitly ensure feasibility of solutions
  - direct constraint handling
    - enforced explicitly during run

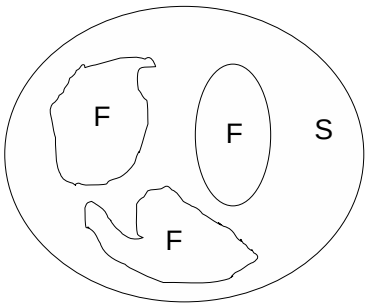
Handling Constraints

- most CSPs are discrete
- two types of COPs
  - discrete COPs (combinatorial optimization)
  - continuous COPs
  - constraint handling basically same

Constraint Types

- inequality / equality constraints
- linear / non-linear constraints

Search Space



Constraint Handling

- three main types
  - indirect constraint handling
  - direct constraint handling
  - mapping constraint handling

Constraint Handling Methods

- indirect constraint handling
  - penalty functions
- direct constraint handling
  - repairing infeasible solution candidates
  - preserving feasibility through special representations and operators
  - seperation of constraints and objectives
- mapping constraint handling
  - using decoder functions

Penalty Functions

Aim: transforming a constrained optimization problem into an unconstrained one by adding / subtracting a value to / from the objective function based on the amount of constraint violation

$$f_p = f \pm p$$

Types of Penalty Functions

- death penalty
- static penalty
- dynamic penalty
- adaptive penalty

Death Penalty

- reject all infeasible solutions
- easiest and computationally most efficient
  - no need to estimate degree of infeasibility
- only advisable if the feasible region is fairly large
  - search may stagnate if the feasible region is very small
- no use of information from infeasible solutions
- a variation of this method assigns zero fitness to infeasibles

Static Penalty

- penalty factors do not depend on current generation number in any way
  - remain constant during whole run
- it may not be a good idea to keep the same penalty factors during the whole run
- penalty factors are problem-dependent

Static Penalty

- extinctive penalties
  - very high penalties to prevent use of infeasibles
- binary penalties
  - d_i is binary: 1 if constraint violated, else 0
- distance based penalties
    - usually use square of Euclidean distance

Dynamic Penalty

- current generation number is involved in the computation of the corresponding penalty factors
  - normally the penalty factors are defined in such a way that they increase over time, i.e. over generations
- difficult to derive good dynamic penalty functions
- require setting of initial values

Adaptive Penalty

- penalty function takes feedback from the search process
- not dependent on initial values
- e.g. penalty for the generation (t + 1)
  - is decreased if all best individuals in the last k generations were feasible
  - is increased if they were all infeasible
  - stays the same if there are some feasible and infeasible individuals tied as best in the population

Adaptive Penalty

- setting the parameters of this type of approach may be difficult
- tries to avoid either an all-feasible or an all-infeasible population
  - more recent constraint-handling approaches pay attention to this issue.

Designing Penalty Functions

- ideal penalty factor cannot be known a priori for an arbitrary problem
  - if penalty too high and optimum lies at the boundary of the feasible region, algorithm will be pushed inside the feasible region very quickly, and will not be able to move back towards boundary
  - if penalty too low, too much search time will be spent exploring the infeasible region because penalty will be negligible with respect to objective function.

Designing Penalty Functions

- appropriate choice of penalty method may depend on:
  - ratio of feasible space to the whole search space
  - topological properties of feasible search space
  - evaluation function
  - number of variables and constraints
- promising results from use of adaptive penalties

Repair Functions

- guarantee feasibility of solution
- infeasibles repaired to closest feasible
- require heuristic
- repair cost
- what to do with repaired solution
  - replace infeasible (Lamarckian)
  - do not replace infeasible (Darwinian / Baldwinian)

Special Representation and Operators

- preserve feasibility
- due to different representation, special operators needed
  - e.g. the TSP with EAs

Seperation of Constraints and Objectives

- constraints and objectives handled seperately
  - multi-objective concepts
  - co-evolution
  - superiority of feasibles

Mapping Constraint Handling

- using decoders to map solutions from the infeasible region into the feasible region
- in some cases, special operators to produce solutions that lie on the boundary of the feasible region have been defined
- main idea is to transform the whole feasible region into a different shape which is easier to explore by the NIH

Example Problem: Multidimensional Knapsack Problem with EAs (MKP)

0/1 Single Knapsack Problem

definition:

- single knapsack of capacity C and n items
- each object has
  - weight w_i
  - profit p_i
- find a vector $x=(x_1,x_2,...x_n)$ where $x_i \in \{0,1\}$ such that:

$$\sum_{i=1}^{n} w_i x_i \leq C \quad \text{for which} \quad P(x) = \sum_{i=1}^{n} p_i x_i \quad \text{is maximized}$$

MKP - Definition

- MKP is a generalization of the 0/1 knapsack problem
- m knapsacks of capacities $c_1,...,c_m$
- n objects with profits $p_1,...,p_n$
- each object has m possible weights
  - object *i* has weight w_{ij} when considered for inclusion in the *j*th knapsack

MKP - Definition

- objective: find a vector $x=(x_1,...,x_n)$ that
  - guarantees no knapsacks are overfilled
  - and yields maximum profit
- solution lies close to boundary of feasible region

$$\max \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_{ij} x_i \leq c_j \quad \text{for} \quad j = 1,2,...,m$$

Real-World Application Examples

- diet problem
  - n different food items
  - m different elements (vitamin A, B, ..., magnesium, ..., calories etc)
  - each element has lower and / or upper limit for intake
  - each food item has cost
  - objective is to minimize cost and adhere to nutritional requirements

Real-World Application Examples

- selecting projects to fund
  - n different projects
  - plan for m years
  - budget determined for each year
  - each project provides a profit
  - objective is to maximize profit and not exceed yearly budgets

MKP - Constraint Handling

- direct search in the complete search space
  - penalty functions
- direct search in the feasible search space
  - clever initialization
  - repair and local search
- indirect search in the feasible search space
  - permutation representation
  - ordinal representation
  - real valued representation
    - random keys
    - weight coding

MKP - Representations

- binary representation
  - if *i*th position is:
    - 1: *i*th item included in **all** knapsacks
    - 0: *i*th item is **not** included in **any** of the knapsacks
  - a string may lead to infeasible solution candidates
- permutation / ordinal representation
  - string shows order to include items
- real valued representation
  - shows a heuristic / random weight for including each item
    - random keys
    - weight coding

Penalty Function

- used with binary representation
- penalty approach proposed by Khuri et al `[Khuri,1994]`
  - allows infeasible strings to join population
  - applies penalty to reduce fitness of infeasible string
  - penalty term gets higher when solution farther away from feasibility

Penalty Function

- new graded fitness function defined - f(x)
  - penalty depends on number of overfilled knapsacks
    - i.e. number of violated constraints

$$f(x) = (\sum_{i=1}^{n} p_i x_i) - s.\max(p_i)$$

where s is no. of overfilled knapsacks

! Does not guarantee all feasible solutions. Check !

Penalty Function

- better penalty function which guides population towards boundary of feasible region – proposed by Gottlieb `[Gottlieb, 1999]`
- new graded fitness function defined - f(x)
  - penalty depends on amount of constraint violation

$$f(x) = (\sum_{i=1}^{n} p_i x_i) - \frac{p_{\max} + 1}{w_{\min}} \max\{CV(x,i) \mid i \in I\}$$

Repair and Local Search

- infeasibles repaired by removing items
  - randomly
  - based on profit-weight ratios
- locally improve found good solutions
- Chu-Beasley, 1998

Permutation Representation

- considers permutations of all items
- a *first-fit* algorithm is used to decode a permutation into a feasible solution
  - starts with the feasible solution $x = (0, \ldots, 0)$
  - considers each item in the order determined by the permutation
  - each corresponding decision variable increased from 0 to 1 if the inclusion of item does not violate any capacity constraints
- all permutation operators may be used
  - uniform order based crossover and swap mutation reported as good

Ordinal Representation

- solution represented by a vector $v = (v1, \ldots, vn)$ with $vk \in \{1, \ldots, n - k + 1\}$ for $k \in \{1, \ldots, n\}$.
- vector mapped to a permutation of the items $\{1, \ldots, n\}$, which is further decoded to a feasible solution via a first-fit heuristic
- example,
  - assume $v = (3, 1, 2, 1)$
  - initially, the ordered list $L = (1, 2, 3, 4)$
  - vector v is decoded by successively removing the elements 3, 1, 4, 2 from L yielding the permutation (3, 1, 4, 2)
- classical operators may be used
- poor performance

Random Key Representation

- based on real-valued vectors $w = (w1, \ldots, wn)$, where each item j is assigned a weight $wj \in [0, 1]$
- decoder sorts all items according to their weights, which yields a permutation in increasing order of weights
- the permutation is decoded via the first-fit heuristic
- two point crossover and gaussian mutation have been used
- performance reported to be inferior to permutation representation

Weight Biased Representation

- solution represented by a vector of real-valued weights
- to obtain phenotype from weight vector:
  - first, the original problem P is temporarily modified to P' by biasing certain problem parameters according to the weights
  - secondly, a problem specific heuristic is used to derive a solution for P'
  - solution to P' is interpreted and evaluated for the original (unbiased) problem P

Weight Biased Representation

- classical positional crossover and mutation operators can be used
- when a suitable biasing scheme and decoding heuristic is used, only feasible candidate solutions are created
- weight-coding approach `[Raidl 1999]`