DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS
FOR COMBINATORIAL OPTIMZATION

Lecture 3

General Methods and Search Algorithms

Marco Chiarandini

Outline

Outline

Methods and Algorithms

A Method is a general framework for the development of a solution algorithm. It is not problem-specific.

An Algorithm (or algorithmic model) is a problem-specific template that leaves some practical details unspecified.
The level of detail may vary:

- minimally instantiated (few details, algorithm template)
- lowly instantiated (which data structure to use)
- highly instantiated (programming tricks that give speedups)
- maximally instantiated (details specific of a programming language and computer architecture)

A Program is the formulation of an algorithm in a programming language.

An algorithm can thus be regarded as a class of computer programs (its implementations)

Solution Methods

- **Exact methods**:
  complete: guaranteed to eventually find (optimal) solution,
  or to determine that no solution exists (eg, systematic enumeration)
  - Search algorithms (backtracking, branch and bound)
  - Dynamic programming
  - Constraint programming
  - Integer programming
  - Dedicated Algorithms
- **Approximation methods**
  worst-case solution guarantee
  `http://www.nada.kth.se/~viggo/problemlist/compendium.html`
- **Heuristic (Approximate) methods**:
  incomplete: not guaranteed to find (optimal) solution,
  and unable to prove that no solution exists
  - Integer programming relaxations
  - Randomized backtracking
  - Heuristic algorithms

Outline

Problem specific methods:

- Dynamic programming (knapsack)

- Dedicated algorithms (shortest path)

Generic methods:

Allow to save development time

Do not achieve same performance as specific algorithms

- Integer Programming (knapsack)

- Search Methods and Constraint Programming
  (constraint satisfaction problem)

Note: In this course we use Search Methods and Constraint Programming, that are generic methods, to learn guidelines in the design of problem-specific construction heuristics.

Knapsack

Knapsack

Given: a knapsack with maximum weight W and a set of n items $\{1, 2, \ldots, n\}$, with each item j associated to a profit p_j and to a weight w_j.

Task: Find the subset of items of maximal total profit and whose total weight is not greater than W.

Bin Packing

One dimensional

Given: A set $L = (a_1, a_2, \dots, a_n)$ of *items*, each with a size $s(a_i) \in (0, 1]$ and an unlimited number of unit-capacity bins $B_1, B_2, \dots, B_m$.

Task: Pack all the items into a minimum number of unit-capacity bins $B_1, B_2, \dots, B_m$.

Related: *cutting stock*

Constraint Satisfaction Problem

- **Input:**
  - a set of variables $X_1, X_2, \dots, X_n$
  - each variable has a non-empty domain D_i of possible values
  - a set of constraints. Each constraint C_i involves some subset of the variables and specifies the allowed combination of values for that subset.

    [A constraint C on variables X_i and X_j, $C(X_i, X_j)$, defines the subset of the Cartesian product of variable domains $D_i \times D_j$ of the consistent assignments of values to variables. A constraint C on variables X_i, X_j is satisfied by a pair of values v_i, v_j if $(v_i, v_j) \in C(X_i, X_j)$.]
- **Task:**
  - find an assignment of values to all the variables $\{X_i = v_i, X_j = v_j, \dots\}$
  - such that it is consistent, that is, it does not violate any constraint

If assignments are not all equally good but some are preferable this is reflected in an objective function.

Outline

Search Methods

- initial state: the empty assignment $\{\}$ in which all variables are unassigned
- successor function: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables
- goal test: the current assignment is complete
- path cost: a constant cost

Two search paradigms:

- search tree of depth n
- complete state formulation: local search

General Purpose Search Algorithms

Search algorithms

tree with branching factor at the top level nd
at the next level $(n-1)d$.
The tree has $n! \cdot d^n$ even if only d^n possible complete assignments.

- CSP is commutative in the order of application of any given set of action. (the order of the assignment does not influence)
- Hence we can consider search algs that generate successors by considering possible assignments for only a single variable at each node in the search tree.

Backtracking search

depth first search that chooses one variable at a time and backtracks when a variable has no legal values left to assign.

Backtrack Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment according to CONSTRAINTS[csp] then
        add {var = value} to assignment
        result ← RECURSIVE-BACKTRACKING(assignment, csp)
        if result ≠ failure then return result
        remove {var = value} from assignment
  return failure
```


Backtrack Search

- No need to copy solutions all the times but rather extensions and undo extensions
- Since CSP is standard then the alg is also standard and can use general purpose algorithms for initial state, successor function and goal test.
- Backtracking is uninformed and complete. Other search algorithms may use information in form of heuristics.

Uninformed Complete Tree Search

- Breadth-first search
- Depth-first search

Informed Complete Tree Search

- Informed search algorithm: exploit problem-specific knowledge
- Best-first search: node that "appears" to be the best selected for expansion based on an evaluation function $f(x)$
- Implemented through a priority queue of nodes in ascending order of f
- See later discussion on A* search

General Purpose Backtracking Methods

1) Which variable should we assign next, and in what order should its values be tried?
2) What are the implications of the current variable assignments for the other unassigned variables?
3) When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

In short, Constraint Programming is a logic programming that express rules and constraints and exploits point 2).

In the general case, at point 1) we use heuristic rules. If we do not backtrack (point 3) then we have a construction heuristic.

1) Which variable should we assign next,
   and in what order should its values be tried?

- Select-Initial-Unassigned-Variable
- Select-Unassigned-Variable
  - most constrained first = fail-first heuristic
    = Minimum remaining values (MRV) heuristic
    (tend to reduce the branching factor and to speed up pruning)
  - least constrained last

  Eg.: max degree, farthest, earliest due date, etc.
- Order-Domain-Values
  - greedy
  - least constraining value heuristic
    (leaves maximum flexibility for subsequent variable assignments)
  - maximal regret
    implements a kind of look ahead

NB: If we search for all the solutions or a solution does not exists, then the ordering does not matter.

Constraint Programming (1)

Types of Variables and Values

- Discrete variables with finite domain:
  complete enumeration is $O(d^n)$
- Discrete variables with infinite domains:
  Impossible by complete enumeration.
  Instead a constraint language (constraint logic programming and constraint reasoning)
  Eg, project planning.

  $$S_j + p_j \leq S_k$$

  NB: if only linear constraints, then integer linear programming
- variables with continuous domains
  NB: if only linear constraints or convex functions then mathematical programming

Constraint Programming (3)

Types of constraints

- Unary constraints
- Binary constraints (constraint graph)
- Higher order (constraint hypergraph)
  Eg, `Alldiff()` `Atmost()`
  Every higher order constraint can be reconduced to binary
  (you may need auxiliary constraints)
- Preference constraints
  cost on individual variable assignments

2) What are the implications of the current variable assignments for the other unassigned variables?

Propagating information through constraints

- Implicit in Select-Unassigned-Variable
- Forward checking (coupled with MRV)
- Constraint propagation
  - arc consistency: force all (directed) arcs uv to be consistent: $\exists$ a value in $D(v)$: $\forall$ values in $D(u)$, otherwise detects inconsistency

    can be applied as preprocessing or as propagation step after each assignment (MAC, Maintaining Arc Consistency)

    Applied repeatedly
  - k-consistency: if for any set of $k-1$ variables, and for any consistent assignment to those variables, a consistent value can always be assigned to any k-th variable.

    determining the appropriate level of consistency checking is mostly an empirical science.

Arc Consistency Algorithm: AC-3

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA*=*red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q*=*green*	R	B	G	R B	R G B	B	R G B
After *V*=*blue*	R	B	G	R	B		R G B

Figure 5.6 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

3) When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Backtracking-Search

- chronological backtracking, the most recent decision point is revisited
- backjumping, backtracks to the most recent variable in the conflict set (set of previously assigned variables connected to X by constraints).

  every branch pruned by backjumping is also pruned by forward checking

  idea remains: backtrack to reasons of failure.

An Empirical Comparison

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV
USA	(> 1,000K)	(> 1,000K)	2K	60
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K
Zebra	3,859K	1K	35K	0.5K
Random 1	415K	3K	26K	2K
Random 2	942K	27K	77K	15K

Median number of consistency checks