Constraint Optimisation Problems

Summary

- Constraint Optimisation
- Cost Network
- Branch and Bound Search
- Bucket Elimination

Constraint Optimisation

Soft Constraint

- Soft Constraints express preferences over variable assignments
- Preferences give different values over variable assignment
- A student can follow only one class at a time (hard constraint)
- A student would like to have no class on Friday (soft constraint)

Global Cost Function

Cost Function and COP

- Optimisation criterion or objective function
- Defined over all the variables
- Constrained Optimisation Problem:
  - find assignment for all variables
  - that satisfies all constraint
  - and optimises the global cost function

COP in practice: Scheduling

Power plant maintainance

- Many power generating units
- Each unit needs to be stopped for preventive maintainance
- Power Plant must not stop generating required energy with remaining units
- Find a schedule (sequence and duration) for single unit maintainance minimising the maintainance cost

COP in practice: Electronic Commerce

Combinatorial Auction

- Combinatorial auction: bidder are allowed to put one bid for a set of items
- Winner determination problem for combinatorial auction
- $S = \{a_1, \cdots, a_n\}$ set of items, $B = \{b_1, \cdots, b_m\}$ set of bids
- $b_i = (S_i, r_i)$, where $S_i \subseteq S$ and r_i is cost paid for bid i
- Find a subset of bids $B' \subseteq B$ such that any two bids in B' do not share any items and $C(B') = \sum_{b_i \in B'} r_i$ is maximised

Combinatorial auction in practice

Regional Fixed Wireless Access

- FWA use of radio to provide last mile connection between users and core telecommunication network
- Used in conuntry with emerging economy: easier and faster to deploy, e.g. Nigeria
- Region based: need to buy licence for regions to roam traffic in that region
- Auctioneer: Government, Bidders: Telecomunication companies, Items: license in each region
- Bid a subset of licences for regions
- Combinatorial: more beneficial to have licenses in "synergic" regions (e.g. adjacent)
- Framework used Nigeria in 2002, business of about 38 billion of USA dollars!

COP and CSP

Max-CSP

- Any CSP can be seen as COP
- Find the assignment that minimises the number of violated constraints: Max-CSP
- Max-SAT: find the assignment that minimises the number of falsified clauses
- When constraint are assigned importance weight
- Goal: minimise the sum of violated constraints

Solving COP

Main approaches

- Search
  - Similar to backtracking
  - Most common approach: branch and bound
- Inference
  - Similar to consistency enforcing approaches
  - Most common approach: dynamic programming

Cost Functions

Definition of Cost Function

- Cost Network: Constraint Network + Cost Function
- Cost Function
  - $X = \{x_1, \cdots, x_n\}$ variables and $\bar{a} = \{a_1, \cdots, a_n\}$ assignments for variables
  - $F_1, \cdots, F_l$ real-valued functional conponents
  - F_i defined over scope $Q_i \subseteq X$

$$F(\bar{a}) = \sum_{j=1}^{l} F_j(\bar{a})$$

- $F_i(\bar{a}) = F_i(\bar{a}[Q_i])$ that is F_i restricted over its scope

Cost Networks

Definition of Cost Networks

- $\mathcal{C} = \{X, D, C_h, C_s\}$
- $\mathcal{R}_c = \{X, D, C_h\}$ Constraint network C_h hard constraints
- $C_s = \{F_{Q_1}, \cdots, F_{Q_l}\}$ soft constraints
- $F_i : \bowtie_{k \in Q_i} D_k \rightarrow \Re^+$
- Aim: find $\bar{a}^*$ such that $\bar{a}$ is a solution for $\mathcal{R}_c$ and $\bar{a}^* = max_{\bar{a}} F(\bar{a})$ (or $\bar{a}^* = min_{\bar{a}} F(\bar{a})$)
- Two variables are linked by a soft constraint if they appear in the scope of a cost function

Example: Cost Network

Example (General Cost Network $\mathcal{CN}$)

- $\mathcal{C} = \{X, D, C_h, C_s\}$
- $X = \{a, b, c, d, f, g\}$
- $C_h = \{\}$
- $C_s = \{F_0(a), F_1(a,b), F_2(a,c), F_3(b,c,f), F_4(a,d,b), F_5(f,g)\}$
- Edges representing soft constraints: $\{< a,b >, < a,c >< a,d >< b,c >< b,f >< b,d >< c,f >< f,g >\}$
- Cost function $C(a,b,c,d,f,g) = F_0(a) + F_1(a,b) + F_2(a,c) + F_3(b,c,f) + F_4(a,d,b) + F_5(f,g)$

Formalisation of Combinatorial Auctions using Cost Networks

Combinatorial Auction

- Variables: b_i Domains: $D_i = \{0, 1\}$
- $b_i = 1$ bid i was selected by the auctioneer
- Hard Constraints: two selected bids can not share any items:
  - $\forall i, j \exists R_{i,j}$ such that $(b_i = 1, b_j = 1) \notin R_{i,j}$
- Soft Constraints: select the bids that maximise the sum of their cost
  - $F_i(b_i) = r_i * b_i$
- Cost function $\sum_{b_i \in B} F_i(b_i)$

Example: A Combinatorial Auction

Example (A Combinatorial Auction $\mathcal{CA}$)

- Items: $\{1, \cdots, 8\}$
- Bids: $\{b_1, b_2, b_3, b_4, b_5\}$
- $b_1 = (\{1,2,3,4\}, 8)$, $b_2 = (\{2,3,6\}, 6)$, $b_3 = (\{1,5,4\}, 5)$, $b_4 = (\{2,8\}, 2)$, $b_5 = (\{5,6\}, 2)$
- $C_h = \{R_{1,2}, R_{1,3}, R_{1,4}, R_{2,4}, R_{2,5}, R_{3,5}\}$
- Graph for constraint network is the same as the graph for cost network (all cost functions are unary)
- Aim: $\max_{b_1,b_2,b_3,b_4,b_5} \sum_{i=1}^{5} b_i * r_i$

Solving COP as a series of CSP

COP as a series of CSP

- Given a cost network $\mathcal{C} = \{X, D, C_h, C_s\}$
- Introduce a cost bound c^i
- Solve the constraint network $\mathcal{R}^i = \{X, D, C_h^i\}$
- Where $C_h^i = C_h \cup H^i$ and $H^i = \sum_{j=1}^{l} F_j \geq c^i$

Solving COP as a series of CSP II

Increasing the cost bound

- Initialise to a low value
- low value depends on the cost functions
  - e.g. if all functions are strictly positive we can set $c^i = 0$
- Increase the cost bound $c^j \geq c^{j-1} \geq \cdots \geq c^1$
- Assume we find a solution s^k for the cost bound c^k and no solution can be found for c^{k+1}
- Then the optimal solution is bounded by $Val(\bar{s}^k) \leq Val(\bar{s}^*) < c^{k+1}$
- $Val(\bar{s}^k) = \sum_i F_i(\bar{s}^k)$ and $\bar{s}^*$ is the optimal solution

Solving COP

Specific approaches

- Using CSP techniques
  - can re-use all efficient techniques seen so far (+)
  - need to solve many CSPs and not clear how many (-)
- Specific techniques are more efficient
  - Branch and Bound
  - Bucket elimination

Branch and Bound

main ideas

- Do a backtracking to find all solution
- Store the current best solution
- Return the best solution found or state problem is inconsistent if no solution exists
- main idea use current best solution to prune useless branches of the search tree

Pruning the search space

Keeping a bound

- Assume we are maximising the cost function
- We can use the current best solution as a lower bound L on any future solution:
  - we can disregard solutions that have a value lower than L
- We need to predict the value of a complete solution given a partial assignment
  - we want to avoid looking further down the tree
- Given a partial instantiation $\bar{a}_i$ we use $f(\bar{a}_i)$
- $f(\bar{a}_i)$ a bounding evaluation function for the possible complete solution
- If $f(\bar{a}_i) \leq L$ we can stop searching along that branch
- $f(\bar{a}_i)$ must be an overestimation to visit all relevant solutions

Controlling the search

Searching technique

- We can use all approaches seen so far for backtracking (Lookahead, Lookback)
- Just need to check the estimated upper bound against the lower bound when selecting a new value
- If minimisation problem exactly same thing:
  - Current best solution is an upper bound U
  - Underestimate the future cost of current partial assignment $f(\bar{a}_i)$
  - Prune if $f(\bar{a}_i) \geq U$

Computing the bounding evaluation function

bounding evaluation function

- Bounding evaluation function is crucial for efficient search
- Need to be easy to compute and as accurate as possible
- If we overestimate too much we never prune
- Simple approach: first choice bounding function
- Similar to forward checking: each constraint is considered independently

$$f_{f_c}(\bar{a}_i) = \sum_j \max_{a_{i+1}, \cdots, a_n} F_j(\bar{a}_i, a_{i+1}, \cdots, a_n)$$

Branch and Bound: Example

Example (Combinatorial Auction)

- Consider the combinatorial auction $\mathcal{CA}$ previously defined
- Consider the order $d = \{b_1, b_2, b_3, b_4, b_5\}$
- Optimal: $\bar{a} = \{0, 1, 1, 0, 0\}$ $F(\bar{a}) = 11$

Dynamic Programming for COP

basic ideas

- Build the solution of a problem incrementally from those of smaller subproblems
- Very convenient for COPs as it exploits the underlying structure of the problem
- Solve subproblems locally and propagate only important information (e.g. counting people along a line)
- Bucket Elimination: dynamic programming procedures to solve COPs

Dynamic programming for COPs: Example

Example (DP for General Constraint Networks)

- Consider the general cost network $\mathcal{CN}$
- Consider the order $d = \{a, c, b, f, d, g\}$
- We want to compute $\max_{a,c,b,f,d,g} F_0(a) + F_1(a, b) + F_2(a, c) + F_3(b, c, f) + F_4(a, d, b) + F_5(f, g)$
- We can manipulate the formula to push maximisation where needed

Bucket Elimination for COPs

Basic Concept

Bucket B_i: a set of constraints that refer to a variable x_i

1. Assign constraints to bucket
2. Process bucket from last variable to first according to a variable ordering
3. Compute optimal tuple propagating values from first variable to last

BE for COPs: Example

Example (Bucket partition)

- Consider the general cost network $\mathcal{CN}$
- Consider the order $d = \{a, c, b, f, d, g\}$
- Buckets: $\{B_a, B_c, B_b, B_f, B_d, B_g\}$
- Partition: $B_g = \{F_5(f, g)\}, B_d = \{F_4(a, d, b)\}, B_f = \{F_3(b, c, f)\}, B_b = \{F_1(b, a)\}, B_c = \{F_2(c, a)\}, B_a = \{F_0(a)\}$

BE: partitioning constraints

Constraint partition

We partition constraints based on variable ordering:

- Put all constraint in a set
- consider variables from last to first according to ordering
- put all constraints in the set that refers to current variable x_i in Bucket B_i
- remove assigned constraints from the set

BE for COPs: Example

Example (Bucket processing)

- Process last buckets: $B_g = \{F_5(f, g)\}$
- $H^g(f) = \max_g F_5(f, g)$, place $H^g(f)$ in bucket B_f
- Process bucket: $B_d = \{F_4(d, b, a)\}$, $H^d(b, a) = \max_d F_4(d, b, a)$, place $H^d(b, a)$ in B_b
- Process bucket: $B_f = \{F_3(f, b, c), H^g(f)\}$, $H^f(b, c) = \max_f(F_3(f, b, c) + H^g(f))$, place $H^f(b, c)$ in B_b
- ...
- Process bucket: $B_a = \{F_0(a), H^c(a)\}$, $M = \max_a(F_0(a) + H^c(a))$

BE: bucket processing

bucket processing

Process bucket based on reverse variable orderings

- Process = Sum all functions and eliminate corresponding variable by maximisation
- This creates a new constraint with scope: all variables mentioned by constraint in this bucket - the variable corresponding to the bucket
- Put new constraint to the highest lower bucket that corresponds to a variable in its scope

BE for COPs: Example

Example (Value propagation)

- Compute $\bar{a_1}^* = \{a^*\}$ $a^* = \arg\max_a(F_0(a) + H^c(a))$
- Compute $\bar{a_2}^* = \{a^*, c^*\}$
  $c^* = \arg\max_c(F_2(a^*, c) + H^b(a^*, c))$
- ...
- Compute $\bar{a_6}^* = \{a^*, c^*, b^*, f^*, d^*, g^*\}$
  $g^* = \arg\max_g(F_5(f^*, g))$

BE: value propagation

value propagation

Propagate values to compute an optimal tuple

- Compute a (partial) tuple that maximise sum of functions of first bucket
- Propagate tuple value to next bucket
- Compute a (partial) tuple that maximises sum of functions given values of propagated tuple from previous bucket

BE for COPs: Example for different ordering

Example (different ordering)

- Consider the cost network $\mathcal{CN}$
- Apply Bucket elimination using $d = \{a, f, d, c, b, g\}$
- Note that functions of 4 variables were created

BE: discussion

Discussion

- After processing buckets (backward phase) we find the optimal value for the COP
- After propagating values (forward phase) we find the optimal assignment for the COP
- Ordering used impacts on size of created functions and therefore on complexity

BE: complexity

Discussion on BE complexity

- The order impacts size of generated functions
- Generated function size equals the number of variables in a bucket minus the bucket variable
- Storing and solving (i.e. maximising) function is exponential in their size
- Thus BE is exponential in the size of the largest bucket.

BE: complexity and induced width

correspondence between BE complexity and induced width

Size of largest bucket equals the induced width of the graph for the given order:

- Functions are recorded when processing all the variables appearing in the bucket
- Variables appearing in a bucket depends on the earlier neighbours according to the ordering
- Such variables should then be connected to represent their relationships in the computation
- This results exactly in the induced graph, and the function size is the induced width

Finding the order that gives the minimum induced width is hard

hard vs soft constraints

hard vs. soft constraints

- We can always encode hard constraints using only soft constraints:
  - Give R_S hard constraints in a maximisation (minimisation) problem
  - Define $F_S(\bar{a}) = 0$ if $\bar{a}$ satisfies R_S
  - $F_S(\bar{a}) = \infty(-\infty)$ otherwise
- However an explicit representation of hard constraints can be more efficient (i.e., enforcing arc consistency etc.)

BE: hard constraints

Incorporating hard constraints

- We can incorporate hard constraints into BE
- Consider in the maximisation only tuples that satisfy the constraints in the bucket:
- Distribute new constraints as with cost function
- For bucket p. R_p join of constraints in the bucket, F^p set of functions in the bucket

$$H^p(t) = max_{\{a_p \mid (t, a_p) \in R_p\}} \sum_{F_i \in F^p} F_i(t, a_p)$$

- Partition constraints into bucket as with cost functions (optional)
- Create new constraints by joining constraints in the bucket and project out the bucket variable (optional)

BE for combinatorial auctions

Example (Solving Combinatorial Auction with BE)

- Consider the combinatorial auction $\mathcal{CA}$
- Apply Bucket elimination using $d = \{b_1, b_5, b_2, b_3, b_4\}$