Scribe Notes: 3/25/13 & 3/27/13
Presenter: Nate Stender
Scribe: Daniel Geschwender

Constraint Optimization: Constraint Processing chapter 13

During these two class periods, Nate discussed the content of Chapter 13 of Dechter's textbook on Constraint Optimization. During, the first period, we covered COPs, the combinatorial auction example, branch-and-bound search, Russian-dolls search, and bucket elimination. During the second period, we covered mini-bucket elimination and its application to branch-and-bound search.

Motivation

- Real life situations cannot always be modeled with hard constraints. Sometimes we need to model degrees of flexibility. Thus, we need to distinguish: Restrictions vs. Preferences.
- Hard constraints must be satisfied, soft constraints should be optimized
- Soft constraints are applicable to planning, scheduling, and design

Motivating Example: Combinatorial Auction

- Bids are placed on groups of items
- No item may be bought twice
- The goal is to select non-overlapping bids that will maximize profit

Bids	**Cost**
$b_1 = \{1,2,3,4\}$	$r_1 = 8$
$b_2 = \{2,3,6\}$	$r_2 = 6$
$b_3 = \{1,4,5\}$	$r_3 = 5$
$b_4 = \{2,8\}$	$r_4 = 2$
$b_5 = \{5,6\}$	$r_5 = 2$

Mutual exclusion constraints:
R_{12}, R_{13}, R_{14}, R_{24}, R_{25}, R_{35}

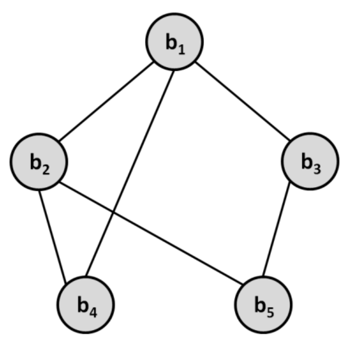

Figure 1: Combinatorial Auction

- Mutual exclusion constraints are placed between bids sharing an item
- Soft constraints are unary, representing the cost if a bid is selected
- The optimal solution is to take $\{b_2,b_3\}$ resulting in a profit of 11

Constraint Optimization Problem (COP)

- A COP consists of a constraint network and a set of cost functions

- Cost function is real valued
- Each functions' scope is a subset of the variables

- A cost function from the combinatorial auction example:

$$F_1(b_1) = \begin{cases} r_1 = 8 \; if \; b_1 = 1 \\ 0 \; if \; b_1 = 0 \end{cases}$$

- Cost functions in the example are unary and Boolean valued. However, in general a cost function may have any arity (i.e., may map any number of tuples). Figure 2 shows an example of a binary cost function F(a,b).
- <u>A cost network</u> is defined by a 4-tuple: (X,D,C_H,C_S)
  - X is the set of variables
  - D is the set of domains
  - C_H is the set of the hard constraints
  - C_S is the set of the soft constraints

a	b	cost
0	0	3
0	1	4
1	0	5
1	1	6

Figure 2: Cost function

Solving a COP as a series of CSPs

- Any COP can be converted into CSPs and solved using standard CSP techniques
- Add a hard constraint to account for the cost of the soft constraints:

$$\sum_{j=1}^{l} F_j \geq C^i$$

- This constraint has a cost bound value, C
- Solve the CSP multiple times, each time increasing the cost bound
- This operation will weed out non-optimal solutions, until only the optimal solution remains
- This technique is expensive because it executes multiple searches

Branch and Bound search

- Branch and bound is a technique that can be added to a backtrack search for solving an optimization problem
- It improves a backtrack search by keeping track of bounds on the optimal solution
  - <u>Lower bound</u> is the cost of the best solution found so far
  - <u>Upper bound</u> is determined from a *bounding evaluation function.*
- The bounding evaluation function will never underestimate the optimal cost[1]
- At each node of the search tree, use the bounding evaluation function to determine upper bounds for each of the child nodes

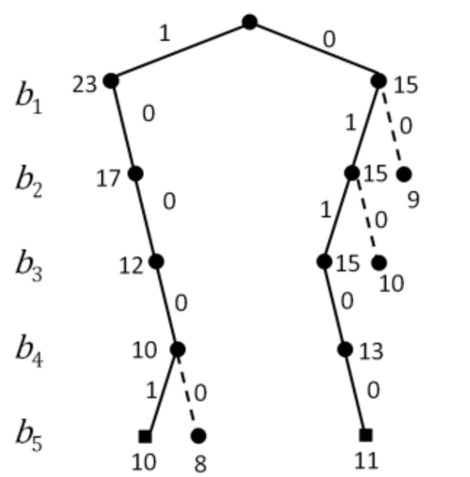

Figure 3: Branch and Bound search

[1] You may remember A* search from the AI course. It combines the cost of the partial solution, g(n), with an evaluation function, h(n), to compute an estimate of the current path, f(n), where f(n)=g(n)+h(n). The tighter the estimate, the smaller the explored search space.

- If the calculated upper bound is less than our lower bound, ignore that path
- Take the path with highest estimated upper bound
- When a complete assignment is found, raise the lower bound to the cost of that assignment
- The search tree generated for the bidding example is shown in Figure 3.
- Branch and bound is beneficial because branches of the search tree can be passed over if the upper bound is less than our lower bound
- Example heuristic for bounding evaluation function:
    - Assume the maximum value for all undetermined cost functions
    - The actual cost couldn't possibly exceed that, but may equal or fall short of that value

Russian-Dolls Search

- This search strategy involves running multiple branch-and-bound searches on increasingly larger problems
    - Begin with the problem consisting of only the last variable and find its optimal solution
    - Each new search adds in another variable, going from the end to the beginning
    - There will be n searches performed, where n is the number of variables in the full problem
- Each search provides an optimal solution for the partial problem, which sets upper bounds for the following searches
- This strategy results in many more searches, but provides a better heuristic, which generally results in more pruning.
- Figure 4 below illustrates the search tree generated by this strategy using the optimal solutions to the small problems ($\{b_5\}$,$\{b_5,b_4\}$,…). Compare the search tree generated by this search and that in Figure 3.

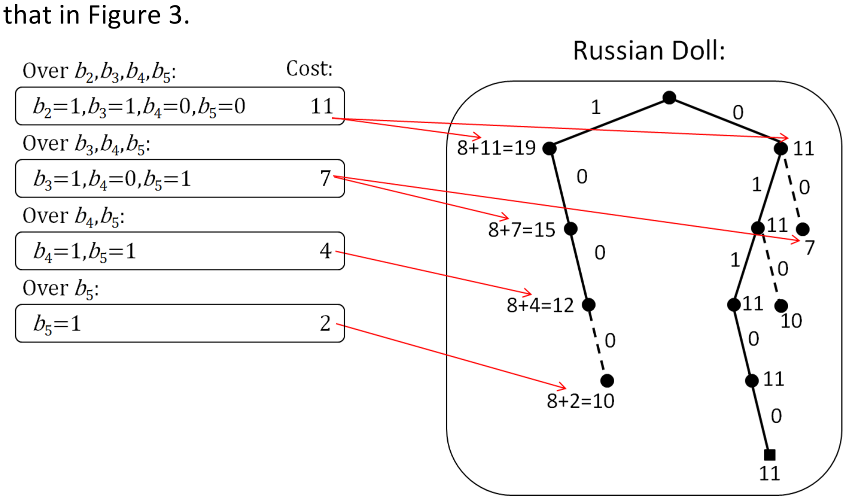

Figure 4: Russian-Doll Search

Improving Branch and Bound

- The branch and bound technique is used often in Operations Research in conjunction with simplex for solving integer programming problems. The integer programming problem is relaxed into a linear programming problem, which is solved by a simplex. The solution of the relaxed problem is then used in a bound and bound search to solve the integer programming problem.

- Best way to improve branch and bound is to improve the bounding function
- A tighter heuristic for the bounding function will result in additional pruning of the search tree
- Bucket elimination can be used with branch and bound

Bucket Elimination for Optimization

- Bucket elimination is an inference algorithm used to solve COPs
- The process is similar to standard bucket elimination but the operations performed on the bucket functions are different.
- Join replaced by summation, projection replace by max (or min)
- The buckets hold cost functions

$$\pi_{x_i} \bowtie R_{ij}$$
$$\downarrow$$
$$\max_{x_i} \sum F_{ai}$$

Figure 5: Operation swap

Question (Fikayo): Why would the min operator be used?

Answer: Some problems are modeled as minimization problems, thus the min operator is used. Others are optimization problems and the max operator is used.

- Processing the buckets forward will give the maximum (minimum) cost value
- Going backwards through the buckets will give you the optimum assignment

Derivation of Bucket Elimination

- The assignments we want will maximize the sum of the cost functions

$$M = \max_{a,c,b,f,d,g} F_0(a) + F_1(a,b) + F_2(a,c) + F_3(b,c,f) + F_4(a,b,d) + F_5(f,g)$$

- The max f unction can be spread out across the function so it is applied only to relevant functions (i.e., the ones that depend on the argument being maximized)

$$M = \max_a F_0(a) + \max_c F_2(a,c) + \max_b F_1(a,b) + \max_f F_3(b,c,f) + \max_d F_4(a,b,d) + \max_g F_5(f,g)$$

- This splitting of the max into separate segments reflects how the buckets are formed
- By generating a function in each bucket, the formula can be further compressed by moving the function to the new lowest bucket
- This derivation of bucket elimination assumes all soft constraints
- To add in hard constraints:
    - First solve hard constraints
    - Prune tuples from the functions that are eliminated by hard constraints
    - Solve for the soft constraints with the remaining tuples

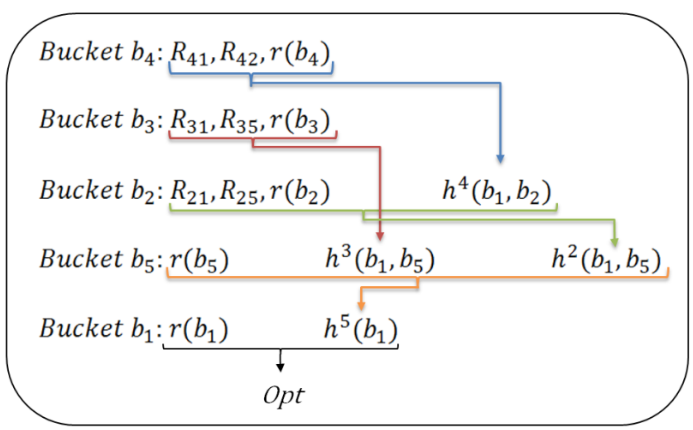

Figure 6: Buckets

Mini-bucket Elimination

- Bucket elimination can be expensive in terms of space: exponential in separator size
- The mini-bucket elimination technique is used to reduce the space requirement while it provides an approximate solution to the problem
- Partitions buckets into smaller buckets
- The resulting functions are of lower arity so they take up less space
- However, this operation results in an approximation of the optimal solution
- Functions in the mini-buckets cannot share variables

Question (Robert): What would happen if a variable was used in multiple functions?

Answer: In the case of hard constraints, you will lose filtering and allow through more solutions. In the case of soft constraints, you will get answers further away from the optimal.

- The mini-bucket elimination will result in an upper bound to the optimal solution
- Smaller mini-buckets are cheaper but less accurate; larger are more costly but closer to optimal
- How your partitions are chosen will affect your results, so a good heuristic is needed
- When processing the buckets in the forward direction, hard constraints may be included in some mini-buckets and left out of others resulting in additional approximation
- The forward direction gives an upper bound

Question (Fikayo): If there are several hard constraints in bucket how do you choose which to ignore?

Answer: They can be chosen in any way, but how they are chosen will affect the results. Thus, good heuristics are important.

Question (Robert): Why can't all the hard constraints be ignored?

Answer: If no hard constraints were taken into account, all costs would be added. The solution would be an upper bound but a very loose one.

- When going in the backward direction, all hard constraints and generated functions are considered
- The backward direction gives a lower bound

Question (DanielG): Does the mini-bucket algorithm offer any guarantee about it bounds?

Answer: No, the algorithm itself does not. The quality of the bounds is entirely dependent on the way the mini-buckets are selected.

- Partitioning the buckets is analogous to moving the max operator within the summation

$$\max\big(Z(x) + Y(x)\big) \leq \max\big(Z(x)\big) + \max(Y(x))$$

Using Mini-Buckets as a heuristic

- Mini-bucket on its own will not provide an optimal solution
- Its bounds can still be useful in a branch and bound search
- The upper bounds generated by mini-bucket will never underestimate the optimal cost
- The bound estimation is monotonic: it will never become less accurate as the search continues
- In the case of the combinatorial auction problem the first choice heuristic performed similarly to mini-bucket
- They may result in different paths
- Mini-bucket provides tighter bounds

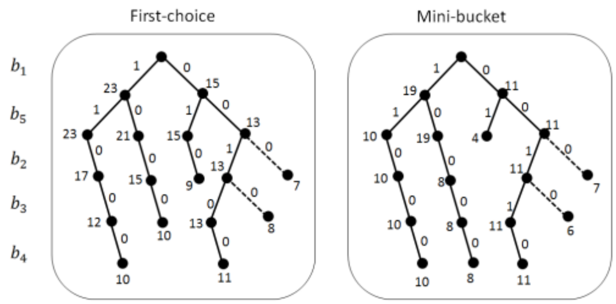

Figure 7: Search tree comparison