Bounded Approximation Algorithms

- Sometimes we can handle NP problems with polynomial time algorithms which are guaranteed to return a solution within some specific bound of the optimal solution
  - within a constant $c$ of the optimal
  - within $\log n$ of optimal, etc.
  - Branch and Bound envelopes

- The bound is the *approximation factor*

- Section 5.4 shows an efficient greedy polynomial time algorithm for Set Cover which is guaranteed to find a solution within $\ln(n)$ of optimal
In clustering we want to find a natural way to group a set of instances into reasonable clusters
- A type of unsupervised learning
- Put similar instances into the same class

Distance function
- Euclidean distance is common, but others can be used

One variation is k-Cluster
- User specifies the number of clusters $k$ to be found
- Often better if $k$ can be discovered automatically
**k-Cluster**

Input: Points $X = \{x_1, \ldots, x_n\}$, integer $k$

Output: A partition of $X$ into $k$ clusters $C_1, \ldots, C_k$

Goal: Minimize the diameter of the clusters

Clusters are spheres all with the same diameter such that all points are enclosed in the spheres

\[
\text{Diameter} = \max_j \max_{x_a, x_b \in C_j} d(x_a, x_b) \text{ for } C_1 \ldots C_k
\]

This problem is exponentially hard – it is in NP

**Figure 9.5** Some data points and the optimal $k = 4$ clusters.
**k-Cluster Approximation Algorithm**

- Choose \( k \) of the data points as cluster centers ("representatives")
- Choose first point arbitrarily
- Choose each succeeding center to be the point furthest from the centers chosen so far (the point furthest from a/its current center)
- Assign all other data points to their closest cluster center – Complexity?

---

Pick any point \( \mu_1 \in X \) as the first cluster center
for \( i = 2 \) to \( k \):  
  Let \( \mu_i \) be the point in \( X \) that is farthest from \( \mu_1, \ldots, \mu_{i-1} \)  
  (i.e., that maximizes \( \min_{j<i} d(\cdot, \mu_j) \))
Create \( k \) clusters: \( C_i = \{ \text{all } x \in X \text{ whose closest center is } \mu_i \} \)

---

**Figure 9.6** (a) Four centers chosen by farthest-first traversal. (b) The resulting clusters.
k-Cluster Approximation Factor

- Definitely clusters, but is max diameter within a factor of optimal?
- Let $x$ be the point furthest from the $k$ cluster centers (i.e. $x$ would be the next center that would have been chosen)
- Let $r$ be the distance of $x$ to the closest $k$
- Then, every point must be within $r$ of its cluster center. Why?
- Thus the maximum diameter of any cluster is $\leq 2r$
- We have $k+1$ points that are all at a distance at least $r$ from each other
- Any partition into $k$ clusters must include 2 of these points in the same cluster with diameter at least $r$. (lower bound)
- Thus, the optimal diameter is between $r$ and $2r$ inclusive and we have an approximation factor of 2

**Figure 9.6** (a) Four centers chosen by farthest-first traversal. (b) The resulting clusters.
Approximation Algorithms

Does this mean we can calculate the optimum once we have our approximation?
Approximation Algorithms

Does this mean we can calculate the optimum once we have our approximation?
- If so, we would have a solution in P and could show that P=NP
- What can we say about the optimum in this type of case?
  - The optimal diameter is within a factor of 2 of the max diameter found by k-Cluster

This gives us another way to deal with NP problems
- Seek polynomial time algorithms that are guaranteed to have a solution within some factor of optimal

This approach can be used for many NP problems
Local Search

- A powerful approach which can be used for any optimization problem
- From a current state try a small move to a neighboring state which improves the overall objective function

```python
let s be any initial solution
while there is some solution s' in the neighborhood of s for which cost(s') < cost(s): replace s by s'
return s
```

- Notion of neighborhood will differ depending on problem
- Neighborhood (Adjustment rate) must also be set
  - Too small – Slow
  - Too big – Jump over good solutions
You own a chocolate shop which produces two types of box chocolates:
- Normal box which gives a $1 profit
- Deluxe box which gives a $6 profit

The variables are the number of boxes produced per day
- \( x_1 \) is the number of boxes of normal chocolate
- \( x_2 \) is the number of boxes of deluxe chocolate

The objective is to set \( x_1 \) and \( x_2 \) so to maximize profit
- \( \max (x_1 + 6x_2) \) Profit = \( x_1 + 6x_2 \)

The constraints are:
- \( x_1 \leq 200 \) Maximum demand of normal boxes per day
- \( x_2 \leq 300 \) Maximum demand of deluxe boxes per day
- \( x_1 + x_2 \leq 400 \) Maximum production capacity
- \( x_1, x_2 \geq 0 \) Can't have a negative number of boxes

Assume fractional to start with, neighbor distance
Could naturally support ILP, non-linear constraints/objectives, etc.
Local Search with TSP

- A legal TSP path is a permutation of the cities: ABCDE
- A close legal neighbor is to swap two cities in the permutation (e.g. ABCDE => ADCBE)
  - How many 2-permutes?
- TSP approach would apply a 2-permute if it shortens the path length and keep doing this until no 2-permute leads to improvement (at a minima)
- Less minima and better solutions if allow 3-permutes
  - At what trade-off?
  - 4-permute, etc. – n-permute is optimal and exponential
- Section 9.3.1 proposes 2-change which drops 2 edges from current path, and adds 2 edges to create a legal path – similar and a bit tighter of a neighborhood than 2-permute: type of 2-opt
  - Other variations – 2-opt, 3-opt, k-opt, etc.,
  - Lots of are other TSP local search approaches: Genetic Algorithms, Simulated Annealing, Ant Colony, etc.
Properties of Local Search

- Can be relatively simple and often one of the most effective algorithms across many applications, regardless of application complexity (non-linear, stochastic, etc.)

- How long will it take?
  - Depends on the number of iterations which is not usually exactly predictable
  - Since each iteration improves the objective function the algorithm works in a constrained space of the problem, but the number of iterations can still be large
  - Often finds a solution relatively efficiently

- Local Optima
  - The good news is the algorithm improves the objective value with each iteration and stops as soon as it can no longer improve
  - The bad news is this could be a local optimum which is not a very good solution
  - Can local search be optimal?
    - Yes when solution is convex (no local optima) and neighborhood size is sufficiently small
Dealing with Local Optima

- The amount of local optima will depend on the problem
  - Some good news coming for many situations
- Note that the search is highly effected by the initial state (usually chosen randomly), since the search will be tied to that initial neighborhood, and often stays close to that neighborhood
- Would if an algorithm has a 50% chance of hitting a local optima for a certain problem
Dealing with Local Optima

- The amount of local optima will depend on the problem
  - Some good news coming for many situations
- Note that the search is highly effected by the initial state (usually chosen randomly), since the search will be tied to that initial neighborhood, and often stay close to that neighborhood
- Would if an algorithm has a 50% chance of hitting a local optima for a certain problem
  - Just run it multiple times with different random start states
  - If chance of hitting a true optima is $p$, then running the problem $k$ times gives probability $1-(1-p)^k$ of finding an optimal solution
- Another approach is to add some randomness to the algorithm and occasionally allow it to move to a neighbor which increases the objective, allowing it to potentially escape local optima
Gradient Descent – Common Powerful Tool

- Slightly change the state in the direction which *maximizes* the improvement in the objective function – steepest gradient
- Usually done by taking the partial derivative of the objective function with respect to the adjustable variables/parameters
- Then we just change the variables in the direction (step size – not too big) of the derivative/gradient which gives the best local change
  - Greedy Search
- We then recalculate the gradient at the new point and repeat until we have reached a minimum (optima)
Brief Inductive Learning Introduction

- Gather a set of labeled examples from some task and divide them into a *training set* and a *test set*
  - e.g. character recognition, medical diagnosis, financial forecasting, document classification, etc.
- Train a learning model (neural network, etc.) on the training set until it solves it well
- The Goal is to *generalize* on novel data not yet seen
- Test how well the model performs on novel data: *Test Set*
- Use the learning system on new examples
Example Application - Heart Attack Diagnosis

- The patient has a set of symptoms - Age, type of pain, heart rate, blood pressure, temperature, etc.
- Given these symptoms in an Emergency Room setting, a doctor must diagnose whether a heart attack has occurred.
- How do you train a machine learning model to solve this problem using the inductive learning model?
- Consistent approach
- Knowledge of machine learning approach not critical
- Need to select a reasonable set of input features
Motivation

- Costs and Errors in Programming
- Our inability to program "subjective" problems
- General, easy-to-use mechanism for a large set of applications
- Improvement in application accuracy - Empirical
Basic Neuron
Generic Neuron and Neurites
Perceptron Node – Threshold Logic Unit

\[ z = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\
0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta 
\end{cases} \]
• Learn weights such that an objective function is maximized.
• What objective function should we use?
• What learning algorithm should we use?
Perceptron Learning Algorithm

\[ z = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\
0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta 
\end{cases} \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>.8</td>
<td>.3</td>
<td>1</td>
</tr>
<tr>
<td>.4</td>
<td>.1</td>
<td>0</td>
</tr>
</tbody>
</table>
First Training Instance

\[
\begin{align*}
\text{net} &= 0.8 \times 0.4 + 0.3 \times -0.2 = 0.26 \\
z &= 1 \quad \text{if} \quad \sum_{i=1}^{n} x_i w_i \geq \theta \\
z &= 0 \quad \text{if} \quad \sum_{i=1}^{n} x_i w_i < \theta
\end{align*}
\]
Second Training Instance

\[ z = 1 \]

\[ net = 0.4 \cdot 0.4 + 0.1 \cdot -0.2 = 0.14 \]

\[ \Delta w_i = (t - z) \cdot c \cdot x_i \]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>0.4</td>
<td>0.1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases} \]
Weight Versus Threshold

\[ X_1 \rightarrow w_1 \]
\[ X_2 \rightarrow w_2 \]
\[ X_n \rightarrow w_n \]
\[ \theta \longrightarrow Z \]

Do you need to adjust Theta? Yes, in most cases

\[ X_1 \rightarrow w_1 \]
\[ X_2 \rightarrow w_2 \]
\[ X_n \rightarrow w_n \]
\[ 1 \rightarrow \]
\[ \theta_{n+1} = -\theta \]
Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate $\alpha$ of 1 and initial weights all 0: $\Delta w_{ij} = \alpha (t_j - z_j) x_i$
- Training set
  
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
<th>Weight Vector</th>
<th>Net</th>
<th>Output</th>
<th>$\Delta W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate $c$ of 1 and initial weights all 0: \[ \Delta w_{ij} = c(t_j - z_j) x_i \]
- Training set
  - 0 0 1 -> 0
  - 1 1 1 -> 1
  - 1 0 1 -> 1
  - 0 1 1 -> 0

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
<th>Weight Vector</th>
<th>Net</th>
<th>Output</th>
<th>$\Delta W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td>0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate $c$ of 1 and initial weights all 0: $\Delta w_{ij} = c(t_j - z_j) x_i$
- Training set

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
<th>Weight Vector</th>
<th>Net</th>
<th>Output</th>
<th>$\Delta W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td>0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
<td>1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate $c$ of 1 and initial weights all 0: $\Delta w_{ij} = c(t_j - z_j)x_i$
- Training set

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
<th>Weight Vector</th>
<th>Net</th>
<th>Output</th>
<th>$\Delta W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>1 1 1 1</td>
<td>3</td>
<td>1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
<td>1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate $c$ of 1 and initial weights all 0: $\Delta w_{ij} = c(t_j - z_j) x_i$
- Training set
  
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
<th>Weight Vector</th>
<th>Net</th>
<th>Output</th>
<th>$\Delta W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td>0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
<td>1 1 1 1</td>
<td>3</td>
<td>1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0</td>
<td>1 1 1 1</td>
<td>3</td>
<td>1</td>
<td>0 -1 -1 -1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0</td>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>1 0 0 0</td>
<td>1</td>
<td>1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
<td>1 0 0 0</td>
<td>1</td>
<td>1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0</td>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
Linear Separability
Linear Separability and Generalization

When is data noise vs. a legitimate exception
Limited Functionality of Hyperplane

Multi-Layer Neural Networks with Backpropagation learning allow us to learn any arbitrary non-linear decision surface.

*** Come to CS 478 to learn more ***
Multi-Layer Generalization
Local Search - Gradient Descent Learning

Error Landscape

TSS: Total Sum Squared Error

Weight Values

0
Local Minima and Neural Networks

- A Neural Network can get stuck in local minima for small networks, but for most large networks (many weights), local minima rarely occur in practice.
- This is because with so many dimensions of weights it is unlikely that we are in a minimum in every dimension simultaneously – almost always a way down.
Application Example - NetTalk

- One of first application attempts
- Train a neural network to read English aloud
- Input Layer - Localist representation of letters and punctuation
- Output layer - Distributed representation of phonemes
- 120 hidden units: 98% correct pronunciation
  - Note steady progression from simple to more complex sounds
Goals of the BYU Neural Networks and Machine Learning Laboratory
http://axon.cs.byu.edu/home.html

- Active PhD and MS students
- Proposal, Extension and Demonstration of improved Learning Models
- Generalization Accuracy
- Speed of Learning, Fault Tolerance
- Models combining the best aspects of Neural Network and symbolic Machine Learning Paradigms
- Various Approaches
- Use applications to drive the research directions
Genetic Algorithms

- For some optimization problems it is desirable to search the space more broadly and not just focus on a good solution based on an initial start state
- Genetic/Evolutionary Algorithms work well in many of these situations
- Accomplishes a more diverse/broad search of the search space than standard local search variations
- Simulates “natural” evolution of structures via selection and reproduction, based on performance (fitness)
- Type of Heuristic Search to optimize any set of parameters
  - A variation on local search using multiple current candidates
Evolutionary Computation/Algorithms
Genetic Algorithms

- Each "Genome" is a current potential solution with a fitness (objective function) score
- Assume we want to solve knapsack with repetition
- If we had 10 possible items what would our genome and fitness look like?
Evolutionary Computation/Algorithms
Genetic Algorithms

- Each "Genome" is a current potential solution with a fitness (objective function) score
- Assume we want to solve knapsack with repetition
- If we had 10 possible items what would our genome and fitness look like?

\[
\text{fitness}(\mathbf{x}) = \begin{cases} 
0 & \text{if } \text{weight}(\mathbf{x}) > W \\
\sum_{i=1}^{\mid\mathbf{x}\mid} \text{value}_i \cdot x_i & \text{Otherwise}
\end{cases}
\]

\[
\begin{array}{ccccccccccc}
1 & 1 & 0 & 2 & 3 & 1 & 0 & 2 & 2 & 1 \\
\end{array}
\]
Evolutionary Computation/Algorithms
Genetic Algorithms

- Populate our search space with initial random solutions
- Use Genetic operators to search the space
- Do local search near the current solutions with mutation
- Do exploratory search with Recombination (Crossover)

```
1 1 0 2 | 3 1 0 2 2 1 (Fitness = 60)
2 2 0 1 | 1 3 1 1 0 0 (Fitness = 72)
```

```
1 1 0 2 1 3 1 1 0 0 (Fitness = 55)
2 2 0 1 3 1 0 2 2 1 (Fitness = 88)
```
Evolutionary Algorithms

- Start with initialized population $P(t)$ - random, domain-knowledge, etc.
  - Each just like an initial parameter setting in local search
- Typically have fixed population size (like beam search), large enough to maintain diversity
- Selection
  - Parent_Selection $P(t)$ - Promising Parents more likely to be chosen based on fitness to create new children using genetic operators
  - Survive $P(t)$ - Pruning of less promising candidates, Evaluate $P(t)$ - Calculate fitness of population members. Could be simple metrics to complex simulations.
- Survival of the fittest - Find and keep best while also maintaining diversity
Evolutionary Algorithm

Procedure EA
\[ t = 0; \]
Initialize Population \( P(t); \)
Evaluate \( P(t); \)
Until Done{ /*Sufficiently “good” individuals discovered or many iterations passed with no improvement, etc.*/
\[ t = t + 1; \]
Parent_Selection \( P(t); \)
Recombine \( P(t); \)
Mutate \( P(t); \)
Evaluate \( P(t); \)
Survive \( P(t); \})
Parent/Survival Selection

- In general want the fittest parents to be involved in creating the next generation
- However, also need to maintain diversity and avoid *crowding* so that the entire space gets explored (local minima vs global minima)
- Most common approach is *Fitness Proportionate Selection* (aka roulette wheel selection)

$$
Pr(h_i) = \frac{\text{Fitness}(h_i)}{|\text{population}|} \sum_{j=1}^{\text{Fitness}(h_j)}
$$

- Everyone has a chance but the fittest are more likely
Genetic Operators

- Crossover variations - multi-point, uniform, averaging, etc.
- Mutation - Random changes in features, adaptive, different for each feature, etc.
- Others - many schemes mimicking natural genetics: dominance, selective mating, inversion, reordering, speciation, knowledge-based, etc.
- Walk through a generation of knapsack example
- Would if we wanted to do Knapsack w/o repetition, fractional knapsack, etc.
Evolutionary Computation Comments

- Much current work and extensions – More detail in 478
- If no better way, can always try evolutionary algorithms, with pretty good results ("Lazy man’s solution" to any problem)
- Many different options and combinations of approaches, parameters, etc.
- Swarm Intelligence – Particle Swarm Optimization, Ant colonies, Artificial bees, Robot flocking, etc.
- More work needed regarding adaptivity of
  - population size
  - selection mechanisms
  - operators
  - representation