Other Metaheuristics: Simulated Annealing, TABU, etc.

Peter G. Anderson,

Computer Science Department

Rochester Institute of Technology,

Rochester, New York

anderson@cs.rit.edu  http://www.cs.rit.edu/
June 11, 2003

Abstract

GAs are not the only metaheuristic available. There are also: exhaustive search, random search, hill-climbing, immune systems, simulated annealing, tabu, ...

And you can make hybrid systems.
Searching for Good Bit-strings

Bit-strings represent **solutions** to problems.

Bit-strings have **fitnesses**

How can we evaluate a large search space?
- Systematic search
- Random search
- Genetic algorithm
- Hill climbing
- Simulated annealing
- Tabu search
Hill Climbing

Assume you have a trial solution
B = some bit string with fitness F.

Examine the **neighborhood** of B for high fitness strings.

A **neighbor of B** differs from B by a very small change.

The neighborhood is a small search space.

Problem: **local optima**
Hill climbing stalls at solutions with all lower-fitness neighbors, but not global optima.
Hill Climbing Pseudocode

```
for (k = 0; k < N; k++) {
    B' = B with k-th bit inverted
    F' = fitness of B'
    if (F' > F) {
        B = B'
        break out of loop
    }
}
```

In case we fail to “break” out of the loop we have failed to improve over B. We may be stuck at a local optimum!
Hill Climbing Code (Main Loop)

fitness = fv(); // evaluate the fitness of p
for( trial = 0; trial < LOOPS; trial++ ) {
  k = random_int( N );
  p[k] = 1 - p[k];
  g = fv();
  if( g > fitness ) {
    fitness = g; // accept new p
  } else {
    p[k] = 1 - p[k]; // reset to previous
  }
}
printf( "Best fitness: %d\n", fitness );
Hill Climbing Variations

- Test all neighbors and move to the greatest improvement.
- Test all neighbors and move to the least improvement.
- Modify the notion of neighbor.
- Use another “mutation” to find a neighbor to test.
Simulated Annealing (SA)

SA sometimes accepts moves to lower fitnesses.

SA uses a temperature

If the system is very hot then SA accepts almost every move

If the system is warm then SA tends to accept improvements

If the system is cold then SA mimics hill-climbing.

This attempts to avoid local optima.

But: do not cool too rapidly
Simulated Annealing

Choose a random neighbor $B'$ of $B$

Compute its fitness, $F'$

Accept the move to solution $B'$ with probability: 
\[
\frac{1}{1 + e^{-\frac{F' - F}{T}}}
\]

$F' - F$ is the fitness improvement.

$T \geq 0$ is the system's temperature
SA Probabilities

$$\Pr[\text{accept } B'] = \frac{1}{1 + e^{-\frac{F' - F}{T}}}$$

Pr[accept $B'$] is near 1.0 if $F' - F$ is large.

Pr[accept $B'$] is near 0.0 if $F' - F$ is small.

The probability varies with temperature.
SA Probability: \[ \frac{1}{1 + e^{-\frac{F' - F}{T}}} \]
Initialize $T$ to hot

while($T > \epsilon$) // $\epsilon > 0$ is small

{ 

$B' = \text{a random neighbor of } B$

$F' = \text{fitness of } B'$

Determine probability $P$

$B = B'$ with probability $P$

if( $F' = \text{goal fitness}$ )

break out of loop

Reduce $T \rightarrow 0$ // slowly

}
Temperature Reduction

“Reduce $T$” very slowly.

\[ T \leftarrow T \cdot 0.999 \]

\[ T \leftarrow T - 0.0001 \]
SA Code (Main Loop)

fitness = fitness_eval();
for( trial = 0; (hero < hero_goal) && (trial < LOOPS); trial++ ) {
    T = T * temperature_decay;
    k = random_int( N );
    p[k] = 1 - p[k]; // invert a random bit
    g = fitness_eval();

    if( drand48() < 1.0 / ( 1.0 + exp( -(g - fitness)/ T ) ) ) {
        fitness = g; // accept the new one
    } else {
        p[k] = 1 - p[k]; // reject the new one
    }
}
printf( "Results are: ....... ");
Tabu Search

As in hill climbing and SA, work with one individual.

Iteratively, test every neighbor, and move to the best one, even if that means going “downtown.”

However, do not change a bit that was changed “recently.”

However, you may change a recently changed bit in case that moves you to a new champion.
Keeping Track of Tabu Moves

Use the variable “trial” for the loop iteration counter.

Whenever a bit, k, is changed in our individual, record:
\[ \text{tabu_list}[k] = \text{trial}; \]

Then, changing bit k is tabu, whenever
\[ \text{trial} - \text{tabu_list}[k] < \text{tabu_tenure}; \]

The tabu_tenure is a parameter do be determined experimentally.
Tabu Search—The Main Loop

do { best_k = -1; best_fit = -9999; trial++; 
   for( k = 0; k < individual_length; k++ ) {
      k_is_tabu = (trial - tabu_list[k]) < tabu_tenure;
      p[k] = 1 - p[k]; // change
      g = fitness(); // evaluate
      p[k] = 1 - p[k]; // repair the change
      if(g > old_hero) || ((g > best_fit) && !(k_is_tabu));
         { best_fit = g; best_k = k; }
   }
   if( best_k > -1 ) {
      p[best_k] = 1 - p[best_k];
      fitness = best_fit;
      tabu_list[best_k] = trial;
   }
} until( stopping condition is met )