Tratamento de problemas $\mathcal{NP}$-difíceis: Algoritmos Genéticos

Eduardo C. Xavier

Instituto de Computação/Unicamp

5 de maio de 2011

Algoritmos Genéticos

- Assim como Simulated Annealing, que simula um processo físico, Algoritmos Genéticos simulam um processo natural.
- Neste caso é um processo biológico, a evolução.
- Por isso algoritmos genéticos são também chamados de algoritmos evolutivos.

Algoritmos Genéticos

Princípio da Evolução:

- Temos uma população de indivíduos competindo para sobreviver (obtém mais comida, foge mais rápido dos predadores etc).
- Indivíduos mais adequados para a competição têm mais chance de viver e gerar filhos. Os filhos herdam as qualidades dos pais.
- Indivíduos sofrem mutações, que ao se mostrarem vantajosa tem mais chance de viver e espalhar tal mutação.

Algoritmos Genéticos

Algoritmos Genéticos:

- A população corresponde à um conjunto de soluções.
- A "competição"é feita com base na função de avaliação de quão boa é a solução.
- Uma solução pode gerar descendentes: membros da sua vizinhança, ou combinando partes de soluções distintas.

Algoritmos Genéticos

Exemplo com o SAT na CNF.

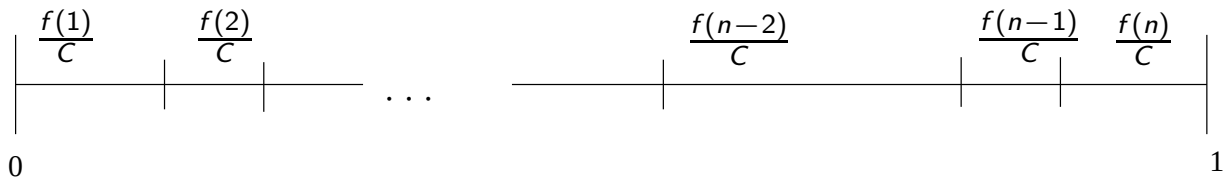
$$F(x) = (\ldots \wedge (x_{12} \vee \neg x_{20} \vee x_3) \wedge \ldots$$

- Achar uma atribuição para as variáveis que faz $F(x)$ ser verdadeira.
- Vamos representar uma solução por um vetor binário.
- Podemos gerar uma população inicial P de 30 soluções, onde em cada solução o bit i recebe $0/1$ com probabilidade $1/2$.
- Dado uma solução t, a função de avaliação $f(t)$ é a quantidade de cláusulas satisfeitas pela solução.
- Vamos manter o tamanho da população sempre em 30.

Algoritmos Genéticos

Exemplo com o SAT na CNF.

- Geramos uma população P' para avaliação.
- P' contém todos os elementos de P mais novas soluções (exemplo 2 vizinhos para cada solução em P).
- A escolha da próxima população P é feita de forma aleatória, mas de tal forma que soluções melhores tenham mais chances de ficar em P.
- Seja $C = \sum_{t \in P'} f(t)$. Cada indivíduo é escolhido para entrar em P com probabilidade $f(t)/C$ (note que o problema é de max.).
- Trinta soluções são escolhidas, uma em cada *round* (note que pode haver repetição de soluções).



Algoritmos Genéticos

```
Algoritmo Genético
    Inicia população P
    Enquanto (não atingir critério de parada) faça
        Gere população P' a partir de P
        P' ← P' ∪ P
        P ← melhores soluções em P'
    Retornar  melhor solução global encontrada.
```

Algoritmos Genéticos

Coisas para decidir:

- Representação das soluções.
- Como gerar população inicial: Soluções aleatórias, ou soluções de outras heurísticas, etc?
- Qual o tamanho de P: é fixo, ou é maior nas primeiras iterações e depois diminui, etc?
- Como avaliar soluções para selecionar as melhores: vai estar relacionada com a função objetivo do problema. Note que podemos penalizar soluções inviáveis ao invés de simplesmente descarta-las.
- Como selecionar soluções: de forma aleatória com preferência para as melhores, ou simplesmente selecionar as melhores?
- Como gerar novas soluções: Vizinhos aleatórios, ou misturar soluções existentes, etc?

Algoritmos Genéticos

- No exemplo do SAT, poderíamos combinar soluções para fazerem parte de P'.
  - Escolher 2 soluções aleatoriamente e escolher de forma aleatória um ponto de quebra.
  - Recombinar gerando duas novas soluções em P'.
    $(1, 0, 1, 0, 0, |1, 1, 1)$ e $(0, 0, 1, 1, 1, |0, 1, 1)$
    Novas
    $(1, 0, 1, 0, 0, |0, 1, 1)$ e $(0, 0, 1, 1, 1, |1, 1, 1)$
- Também podemos gerar alguns vizinhos de cada solução (exemplo: invertendo um bit) para fazer parte de P'.

Algoritmos Genéticos

Exemplo com TSP.

- Temos um grafo completo com pesos nas arestas.
- Devemos achar um ciclo de custo mínimo que passa por cada vértice exatamente uma vez.
- Podemos representar uma solução por um vetor permutação dos n vértices:

$$(1, 45, 10, \ldots, 9, 36)$$

Neste exemplo saimos de 1 para 45, depois para 10, etc., até chegar ao vértice 36 e voltamos para 1.

Algoritmos Genéticos

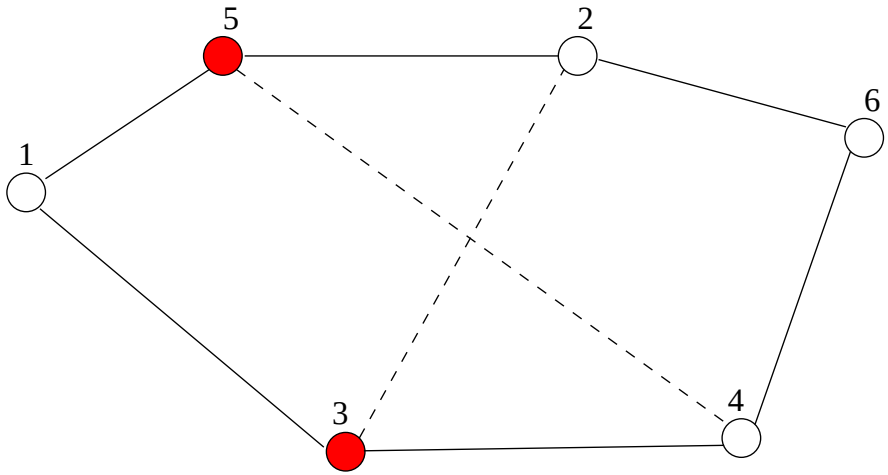
Exemplo com TSP.

- Vamos assumir novamente que o tamanho da população P é 30.
- A população inicial pode ser obtida escolhendo-se 30 permutações aleatórias de 1 até n, ou pode-se usar soluções obtidas por algumas heurísticas gulosas rápidas.
- A função de avaliação nada mais é do que o custo do ciclo.
- Devemos agora gerar uma população P' para avaliação.

Algoritmos Genéticos

Exemplo com TSP.

- Uma idéia é gerar vizinhos para cada $s \in P$ com a operação 2-opt.
- Isto significa a troca de duas arestas:



- $s = (1, 5, 2, 6, 4, 3)$ e vizinho é $(1, 5, 4, 6, 2, 3)$.
- Note que a operação escolhe duas posições i e j com $(n \geq j \geq i + 3)$ do vetor e inverte os vértices entre estas posições.

Algoritmos Genéticos

Exemplo com TSP.

- Uma outra idéia para gerar elementos em P' é tentar juntar duas soluções s_1 e s_2 de P.
- Podemos copiar o início de uma solução s_1 e depois copiar os vértices que faltam na ordem em que aparem em s_2.

$$s_1 = (4, 5, 2, 1, 7, 3, 6)$$

e

$$s_2 = (4, 7, 2, 3, 1, 5, 6)$$

sorteamos uma posição aleatória (exe: 3):

$$(4, 5, 2, \ldots) \rightarrow (4, 5, 2, 7, 3, 1, 6)$$

Algoritmos Genéticos

Exemplo com TSP.

- Podemos gerar uma população P' com 90 elementos.
- Depois escolhemos as 30 melhores soluções dentre P e P' para ser a nova população corrente P.
- Podemos ainda selecionar de forma aleatória (como no SAT), mas dando prioridade as melhores soluções.
- Uma outra alternativa muito comum na prática é escolher a nova população P apenas de P'.

Algoritmos Genéticos

Finalizando a idéia básica é:

1. Manter população P de melhores soluções.
2. Gerar conjunto P' de novas soluções a partir de P.
3. Selecionar como novo P as melhores soluções em P'.

Os passos acima são repetidos um determinado número de iterações, ou até que pouca melhora ocorra durante o processo.

- Existem várias outras possibilidades de incremento do método básico, como por exemplo incluir soluções aleatórias em P', mudar os parâmetros durante a execução, etc.

Heurísticas de Busca Local

- Todas as heurísticas vistas tentam de uma forma ou de outra resolver o seguinte problema:
  - Achar a melhor solução possível evitando ótimos locais.
- Busca Tabu mantém uma memória de soluções (ou operações) já realizadas, para que novas soluções (mesmo que piores a princípio) sejam avaliadas.
- GRASP: Gera soluções iniciais aleatorias gulosas recomeçando em diferentes partes do espaço de soluções.
- Simulated annealing mantém um parâmetro aleatório, fazendo que no início soluções diversas possam ser avaliadas. Mas com o tempo chegamos a um ótimo local.
- Algoritmos Genéticos mantém uma população para avaliação. A diversificação se da pelas diferentes soluções na população, pois novas soluções são geradas olhando-se a população como um todo (cruzando soluções etc.).