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

Flávio K. Miyazawa
Eduardo C. Xavier

Instituto de Computação/Unicamp

29 de abril de 2011

GRASP
Greedy Randomized Adaptive Search Procedures

Duas fases:

1. Algoritmo guloso $\times$ Construção aleatória
   - Construção aleatória
     - Soluções com alta variabilidade.
     - Baixa qualidade de soluções.
   - Algoritmo Guloso
     - Soluções de boa qualidade.
     - Baixa ou nenhuma variabilidade nas soluções.
   - GRASP: Explorar vantagens das duas estratégias.
2. Busca Local
   - Aplicar busca local nas soluções da primeira fase.

GRASP

GRASP: Greedy Randomized Adaptive Search Procedures

- Insere aleatoriedade na geração das soluções gulosas.
  - Cada solução é formada por elementos/componentes.
  - Elementos/componentes são rankeados de acordo com valor acrescido na solução.
  - Elementos são inseridos de forma gulosa/aleatória para formar solução inicial.
- Para cada solução inicial, aplica método de busca local.
- Guardar a melhor solução encontrada durante sua execução.

GRASP

```
GRASP-Simplificado      (* problema de minimização *)
    S ← ∅;
    S* ← ∅;
    Enquanto (não atingir critério de parada) faça
        S ← Solução-Gulosa-Aleatória();
        S' ← Busca-Local(S);
        Se custo(S') < custo(S*) então
            S* ← S'
    fim-enquanto
    Retornar  S*.
```

Possíveis implementações das subrotinas:

Condições de parada

- Número de iterações limitado a um valor máximo
- Quando atingir limite de tempo de CPU
- Quando melhor solução não for atualizada por certo número de iterações

Possíveis implementações das subrotinas:

Solução-Gulosa-Aleatória
$S \leftarrow \emptyset$
Enquanto S não é solução viável **faça**
$L \leftarrow$ Construa-Lista-Restrita-de-Candidatos(S)
$e \leftarrow$ Escolha-Gulosa-Aleatória(L)
$S \leftarrow$ Insere-Novo-Elemento(S, e)
Devolva S

Construa-Lista-Restrita-de-Candidatos:

- A cada iteração temos uma solução parcial S.
- Seja $f(S, e)$ valor da solução parcial S acrescida de elemento e.
- Seja $E = (e_1, \ldots, e_m)$ elementos/componentes que podem ser inseridos em S ordenados:

$$f_{\min} = f(S, e_1) \leq \ldots \leq f(S, e_m) = f_{\max}$$

Construa-Lista-Restrita-de-Candidatos (Minimização)

Por qualidade mínima

1. Seja $\alpha \in [0, 1]$.
2. $t \leftarrow \max\{i : \ f(S, e_i) \leq f_{\min} + \alpha \cdot (f_{\max} - f_{\min})\}$
3. $L \leftarrow (e_1, \ldots, e_t)$
4. Devolva L

Pelos k melhores elementos:

1. Seja k tamanho máximo para lista restrita de candidatos
2. $L \leftarrow (e_1, \ldots, e_k)$
3. Devolva L

Escolha-Gulosa-Aleatória:

Possíveis algoritmos para Escolha-Gulosa-Aleatória(L):

- Seja a LRC, $L = (e_1, \ldots, e_k)$.

Escolha-Gulosa-Aleatória(L) (Minimização)

Uniforme

- Com probabilidade $\frac{1}{|L|}$, devolva $e \in L$.

Ordem

- $bias(e_i) \leftarrow \frac{1}{i}$ para $i = 1, \ldots, k$ (indicador de preferência).
- Defina $p(e_i)$ probabilidade de obter e_i proporcional a

$$\frac{\text{bias}(e_i)}{\sum_{i=1}^{k} \text{bias}(e_i)}$$

- Com probabilidade $p(e_i)$, devolva $e \in L$.

Exemplo: TSP

- Lista de elementos E que compõem uma solução são arestas.
- Uma solução parcial S será composta por caminhos sem vértices em comum.
- Quando tivermos um único caminho sobre todos os vértices temos uma solução.
- Lista-Restrita-de-Candidatos
  - Elementos candidatos a serem inseridos em S são arestas que quando adicionadas a S garantem:
    - Todos os vértices em S terão grau no máximo 2.
    - Ou seja, são arestas que ligam vértices de grau ≤ 1.
  - $f(S, e)$ é o custo dos caminhos quando inserido e.

Path Relinking

Idéia:

- Sejam S e T duas soluções boas.
- Suponha que fazemos movimentos que partem de S para T.

$$S = S_1 \to S_2 \to \ldots \to S_k = T$$

- Possivelmente, neste caminho podemos obter soluções melhores que obtêm características boas de ambas soluções

Path Relinking

Exemplo: CNF-SAT com pesos nas cláusulas.

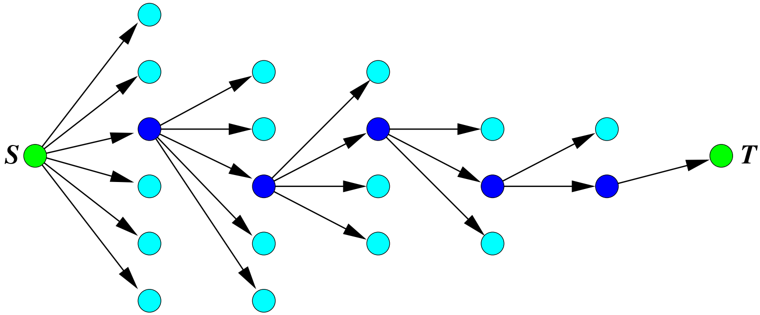
- Duas soluções S e T compostas por atribuição $1/0$ para variáveis $(x_1, \ldots, x_n)$.
- Diferença simétrica $\Delta(S, T)$ são variáveis com valores distintos em S e em T.
- Podemos mudar uma variável por vez de S até solução ficar igual a T.

Path Relinking

- Denote por $\Delta(S, T)$ a diferença simétrica de S e T
- S^e denota solução vizinha de S_j com transformação $e \in \Delta(S, T)$

Path-Relinking(S, T)
  $S^* \leftarrow S$ (manter a melhor solução)
  $S_0 \leftarrow S$
  $j \leftarrow 1$
  Enquanto $j < |\Delta(S, T)|$ **faça**
    $S_j \leftarrow$ solução em $\{S^e : e \in \Delta(S_{j-1}, T)\}$ de custo mínimo
    Se $c(S_j) < c(S^*)$ **então** $S^* \leftarrow S_j$
    $j \leftarrow j + 1$
  Devolva S^*

Path Relinking



Forward Path Relinking: S é uma solução melhor que T

Backward Path Relinking: S é uma solução pior que T

GRASP with Path Relinking - Minimização

- Manter um pool P das melhores soluções
- A cada iteração do GRASP (após computar S' da busca local):
  1. Escolher uma solução T em P.
  2. Realizar Path-Relinking entre S' e T.
  3. Atualizar melhor solução S^* e pool caso necessário.

Ex.: MaxSat (Festa, Pardalos, Pitsoulis, Resende'06)

Comparação: GRASP $\times$ GRASP+Path Relinking:

Probabilidade de se alcançar valor de uma solução pelo tempo