GRASP
Greedy Randomized Adaptive Search Procedures

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

GRASP

GRASP: Greedy Randomized Adaptive Search Procedures

- Em cada iteração, aplica método de busca local
- Insere aleatoriedade na geração das soluções iniciais
- Multi-Start Local Search + Soluções Iniciais guiadas por processo Guloso-Probabilístico
- Cada solução é formada por elementos/componentes
- Cada elemento/componente é rankeado de acordo com uma função gulosa
- Guarda a melhor solução encontrada durante sua execução

GRASP

GRASP Simplificado - Minimização

13. $S^+ \leftarrow \emptyset$ (mantém atualizado a melhor solução encontrada)
14. Enquanto não atingir condição de parada faça
15. $\quad S \leftarrow$ *Solução-Gulosa-Aleatoria*()
16. $\quad S' \leftarrow$ *Busca-Local*(S)
17. $\quad$ Se $c(S') \leq c(S^+)$ então
18. $\quad\quad S^+ \leftarrow S'$
19. Devolva 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

1. $S \leftarrow \emptyset$
2. Enquanto S não é solução
3. $\quad L \leftarrow$ *Construa-Lista-Restrita-de-Candidatos*(S)
4. $\quad e \leftarrow$ *Escolha-Gulosa-Aleatória*(L)
5. $\quad S \leftarrow$ *Insere-ou-Adapte-Novo-Elemento*(S, e)
6. Devolva S

Construa-Lista-Restrita-de-Candidatos:

- Seja $f(S, e)$ valor da função gulosa sobre acréscimo do elemento e em S
- Seja $E = (e_1, \ldots, e_m)$ elementos/componentes candidatos para serem inseridas (e alteradas) em S tal que

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

CONSTRUA-LISTA-RESTRITA-DE-CANDIDATOS (Minimização)

min max α:

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

Por cardinalidade k:

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

Escolha-Gulosa-Aleatória:

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

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

ESCOLHA-GULOSA-ALEATÓRIA(L) (Minimização)

Uniforme

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

Ordem

1. $bias(e_i) \leftarrow \frac{1}{i}$ para $i = 1, \ldots, k$ (indicador de preferência).
2. Defina $p(e_i)$ probabilidade de obter e_i proporcional a $bias(e_i)$
3. Com probabilidade $p(e_i)$, devolva $e \in L$.

Outras

- Baseado no peso dos elementos.
- Baseado em $f(S, e)$.
- Combinação de regras.
- etc.

Path Relinking

Idéia:

1. Sejam S e T duas soluções
2. Suponha que fazemos movimentos que partem de S para T.

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

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

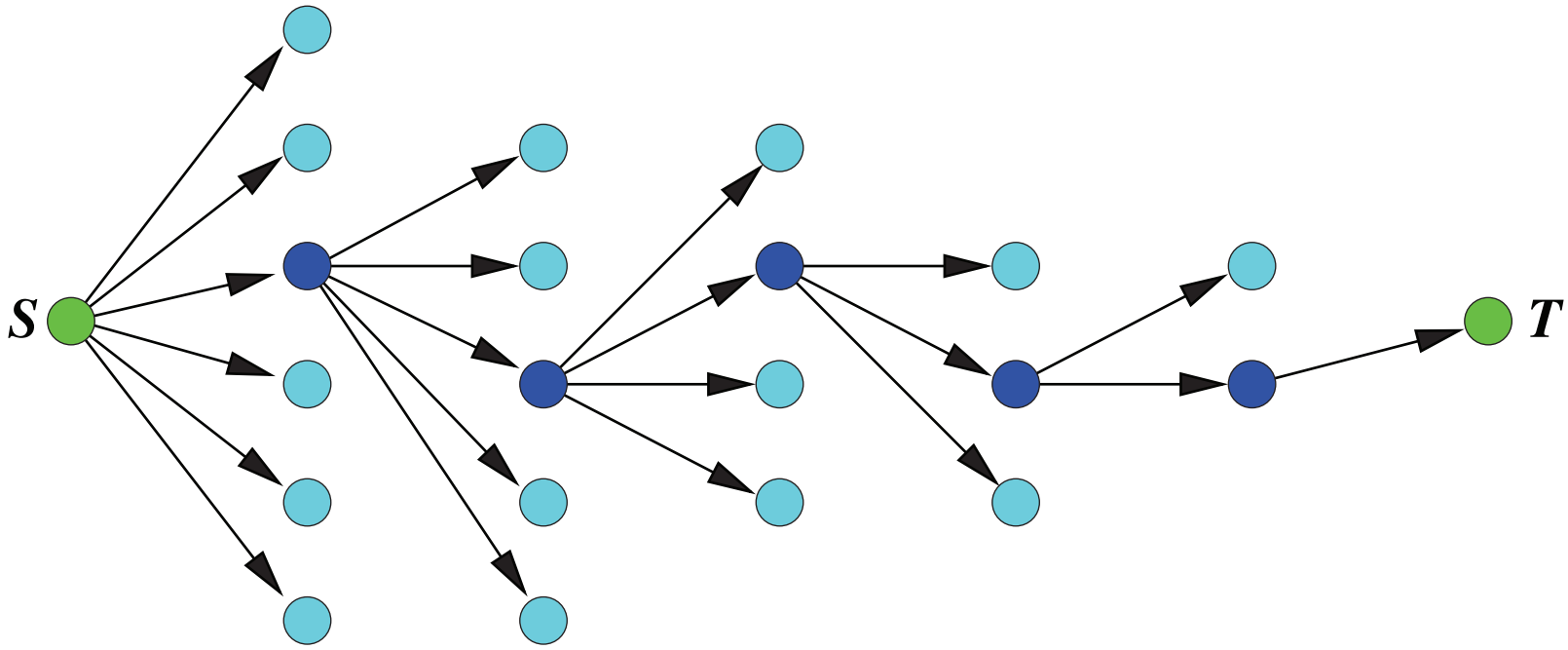
Path Relinking

Denote por $\Delta(S, T)$ a diferença simétrica de S e T

PATH-RELINKING(S, T)

1. $j \leftarrow 1$
2. $S_j \leftarrow S$
3. $S^+ \leftarrow S_j$ (manter a melhor solução)
4. Enquanto $\Delta(S_j, T) \neq \emptyset$
5. Seja S_j^e solução vizinha de S_j correspondente a $e \in \Delta(S_j, T)$
6. Seja $S_{j+1} \in \{S_j^e : e \in \Delta(S_j, T)\}$ com custo mínimo
7. Se $c(S_{j+1}) < c(S^+)$ então $S^+ \leftarrow S_{j+1}$
8. $j \leftarrow j + 1$
9. Devolva S^+

Path Relinking



Forward Path Relinking: *S* é uma solução melhor que *T*

Backward Path Relinking: *S* é uma solução pior que *T*

Back and forth Path Relinking: Busca de *S* para *T* e depois de *T* para *S* (custo computacional duplicado, mas possivelmente melhora marginal)

GRASP with Path Relinking - Minimização

- Manter um pool $\mathcal{P}$ das melhores soluções
- Seja S_j a solução obtida pelo GRASP pela busca local da j-ésima iteração.
- No fim da iteração j do GRASP, faça
  1. $T \leftarrow$ Escolha-Solução-Destino$(\mathcal{P}, S')$
  2. $S' \leftarrow$ Path-Relinking(S_j, T)
  3. Se $c(S') < c(S^+)$ então $S^+ \leftarrow S'$
  4. Atualize-Pool$(\mathcal{P}, S')$

Alternativas de implementação da rotina: Escolha-Solução-Destino

Escolha-Solução-Destino($\mathcal{P}, S'$) - Minimização

- Escolha uma solução $T \in \mathcal{P}$ com probabilidade uniforme
- Escolha uma solução $T \in \mathcal{P}$ com probabilidade proporcional a $|\Delta(T, S')|$

Possível implementação da rotina: Atualize-Pool

- Manter boas soluções destino no pool, mantendo diversidade
- Remover soluções piores ou parecidas da que está inserindo

Possível implementação para
Atualize-Pool($\mathcal{P}, S'$) - Minimização

1. Se $|\mathcal{P}| < MaxPool$ então
2. $\quad \mathcal{P} \leftarrow \mathcal{P} \cup S'$
3. Senão
4. $\quad \mathcal{Q} \leftarrow \{S \in \mathcal{P} : \ c(S) \geq c(S')\}$
5. $\quad$ Se $\mathcal{Q} \neq \emptyset$ então
6. $\quad\quad$ Seja $Q \in \mathcal{Q}$ com $|\Delta(Q, S')|$ mínimo
7. $\quad\quad \mathcal{P} \leftarrow \mathcal{P} - Q + S'$

Exemplo: Max-Sat com pesos nas cláusulas

Lista-Restrita-de-Candidatos:

A cada solução parcial, calcula para cada variável X_i o peso total das novas cláusulas satisfeitas quando $X_i = 1$ ou $X_i = 0$.

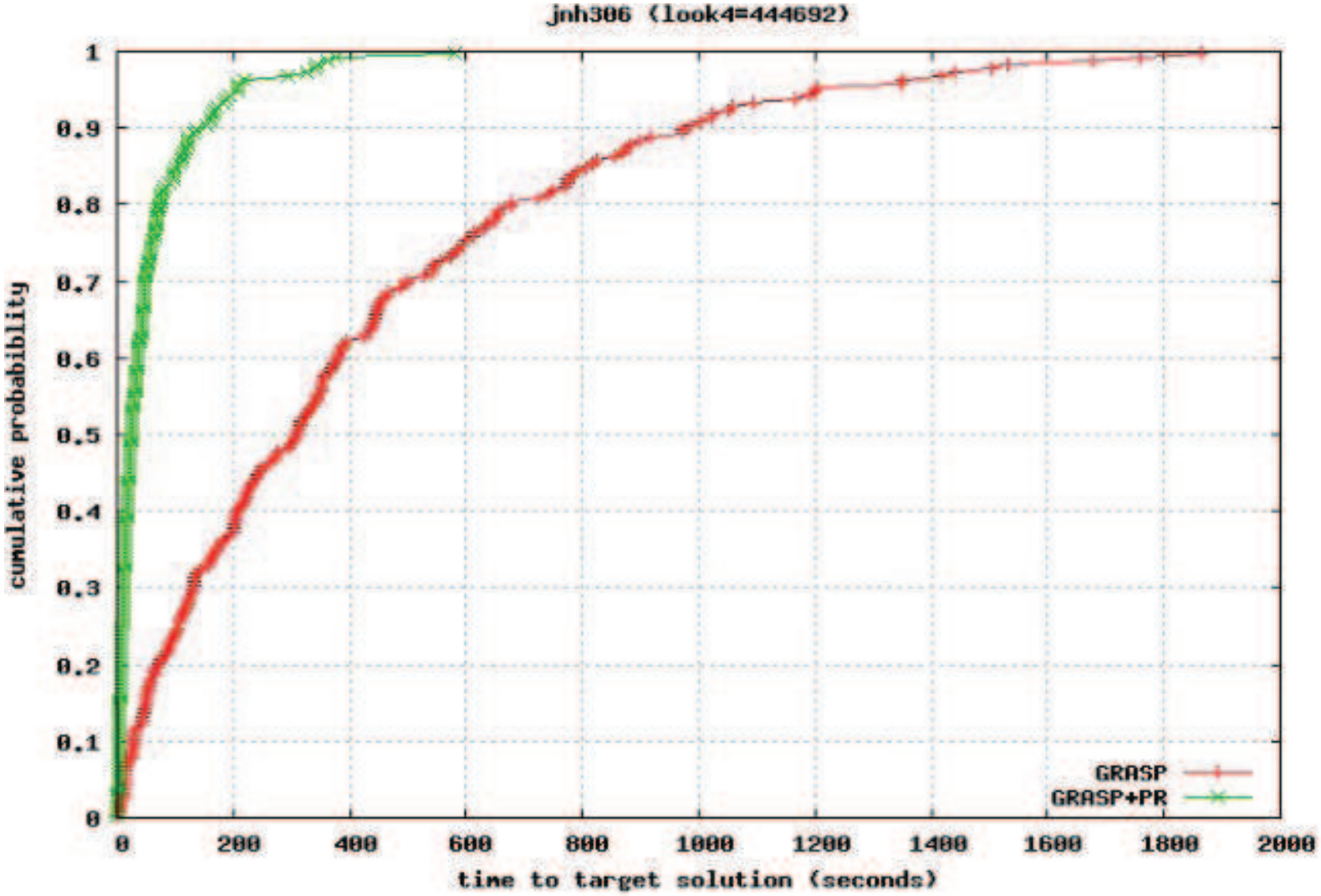
Diferença simétrica

$$\Delta(X, Y) = \{i : \ X_i \neq Y_i, \quad i = 1, \ldots, n\}$$

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

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

Probabilidade de se alcançar valor de uma solução pelo tempo



GRASP

Exercícios faça heurísticas GRASP para os seguintes problemas:

1. Caixeiro Viajante: TSP
2. Corte de Peso Máximo: MaxCut
3. Satisfatibilidade de Peso Máximo: MaxSat