

Metaheurísticas

Prof. Carlos Alberto de Araújo Padilha

12 de Abril, 2018

- 1 Introdução
- 2 Metaheurísticas Baseadas em Solução Única
 - Simulated Annealing
 - Busca Tabu
 - Variable Local Search
 - Iterated Local Search
- 3 Metaheurísticas Baseadas em População
 - Algoritmos Genéticos
- 4 Práticas

- 1 Introdução
- 2 Metaheurísticas Baseadas em Solução Única
 - Simulated Annealing
 - Busca Tabu
 - Variable Local Search
 - Iterated Local Search
- 3 Metaheurísticas Baseadas em População
 - Algoritmos Genéticos
- 4 Práticas

- Uma metaheurística é um algoritmo desenvolvido para resolver, de maneira aproximada, problemas de otimização complexos sem a necessidade de uma grande adaptação para cada problema.
- Problemas de otimização complexos são aqueles que não podem ser resolvidos de forma **ótima** ou por qualquer método exato em um tempo **razoável**.
- Encontrados em áreas indo das finanças até ao gerenciamento de produção e engenharia.
- As metaheurísticas são geralmente utilizadas onde faltam soluções heurísticas satisfatórias.

- De uma forma geral, as metaheurísticas possuem as seguintes características:
 - Inspiradas na natureza, ou seja, são baseadas em princípios da física ou biologia;
 - Usam componentes estocásticos;
 - Não fazem uso de derivada;
 - Possuem parâmetros que precisam ser ajustados ao problema.

- O bom desempenho de uma metaheurística em um problema de otimização dependerá se ela conseguir balancear a **exploração** e a **intensificação**.
- Exploração
 - Consiste em sondar diferentes porções do espaço de busca visando identificar soluções promissoras e diversificar a busca, a fim de evitar ficar preso em um ótimo local.
- Intensificação
 - Sonda uma porção limitada do espaço de busca com a esperança de melhorar a solução que já se tem em mãos.
- Segundo [Birattari et al. 2001], a forma particular como cada algoritmo alcança esse balanço é o que mais as diferencia.

- Há vários aspectos que podem ser usados para classificar esses algoritmos, como:
 - Processo de busca;
 - Uso de memória;
 - Vizinhança;
 - Quantidade de soluções mantidas para a próxima iteração/geração;
 - Maneira de fugir de mínimos/máximos locais.
- Entretanto, a classificação mais utilizada na literatura é separá-las entre metaheurísticas baseadas em **solução única** e baseadas em **população**.

- 1 Introdução
- 2 Metaheurísticas Baseadas em Solução Única
 - Simulated Annealing
 - Busca Tabu
 - Variable Local Search
 - Iterated Local Search
- 3 Metaheurísticas Baseadas em População
 - Algoritmos Genéticos
- 4 Práticas

Metaheurísticas Baseadas em Solução Única

- Ao contrário daquelas baseadas em população, inicializam o processo de busca com uma solução inicial e, então, se movem no espaço de busca descrevendo uma trajetória.
- Os principais algoritmos que se enquadram nessa categoria são:
 - *Simulated Annealing*
 - Busca Tabu
 - GRASP
 - *Variable Neighborhood Search*
 - *Iterated Local Search*
 - Outras variantes
- Aqui veremos um pouco mais sobre *Simulated Annealing*, Busca Tabu, *Variable Neighborhood Search* e *Iterated Local Search*.

Simulated Annealing

- O algoritmo Simulated Annealing ou têmpera simulada, em português, foi proposto por [Kirkpatrick, Gelatt e Vecchi 1983] e [Cerny 1985].
- Tem como base o processo metalúrgico de têmpera para obter um estado sólido de energia mínima onde a liga metálica é bastante forte.
- O processo de têmpera consiste em inicialmente **eleva**r a temperatura de um sólido em banho térmico, fazendo com que as partículas do material se distribuam aleatoriamente numa fase líquida, e então, **gradualmente resfriando** o objeto até a temperatura ambiente, de maneira que todas as partículas fiquem arranjadas em um estado de **baixa energia**, onde a solidificação acontece.

Simulated Annealing

- *Simulated Annealing* transpõe o processo de têmpera para resolver problemas de otimização:
 - A função objetivo do problema f que, de forma similar à energia do material, é minimizada.
 - Usa um parâmetro T que simula a temperatura do objeto.



Simulated Annealing

- O algoritmo começa com uma solução inicial s , escolhida de forma aleatória ou através do uso de alguma heurística, e inicializando o parâmetro de temperatura T .
- Então, a cada iteração, uma solução s' é escolhida aleatoriamente na vizinhança de s que é a solução atual do problema.
- A solução atual do problema é atualizada dependendo dos valores da função objetivo para s e s' ($f(s)$ e $f(s')$) e também do valor atual de T .
- Caso $f(s') \leq f(s)$ (considerando um problema de minimização), o valor de s é atualizado com s' .
- Por outro lado, se $f(s') > f(s)$, s' ainda poderá ser aceita como solução atual com uma probabilidade $p(f(s), f(s'), T) = \exp(-\frac{f(s') - f(s)}{T})$ (algoritmo de Metropolis [Metropolis et al. 1953]).

Simulated Annealing

- Esse processo por um número máximo de iterações (S_{Amax}) para cada temperatura T .
- A temperatura T é reduzida lentamente durante o processo de busca usando um fator de resfriamento α , tal que $T_k = \alpha * T_{k-1}$, sendo $0 < \alpha < 1$.
- Isso quer dizer que, no início da busca, a probabilidade do algoritmo aceitar soluções de piora é alta e, então, diminui gradativamente com o passar das iterações.

Simulated Annealing

Input: Uma solução inicial s escolhida aleatoriamente

Input: Temperatura inicial T

Input: Número máximo de iterações S_{max}

Input: Parâmetro de ajuste de temperatura α

$IterT = 0$

while critério de parada não é atingido **do**

while $IterT < S_{max}$ **do**

$IterT = IterT + 1$

 Selecione aleatoriamente uma solução s' da vizinhança $N(s)$

if $f(s') \leq f(s)$ **then**

$s = s'$

end

else

 Gere um valor aleatório v entre 0 e 1

 Calcule a probabilidade de aceitação

$$p(f(s), f(s'), T) = \exp\left(-\frac{f(s') - f(s)}{T}\right)$$

if $v \leq p(f(s), f(s'), T)$ **then**

$s = s'$

end

end

end

$IterT = 0$

$T = \alpha * T$

end

Output: A melhor solução s encontrada

Simulated Annealing

- Na literatura há diversos esquemas para realizar o resfriamento da temperatura, como, por exemplo, os resfriamentos aditivos e multiplicativos.

- Aditivos

- Aditivo linear: $T_i = T_0 - i * \frac{(T_0 - T_N)}{N}$
- Aditivo quadrático: $T_i = T_N + (T_0 - T_N) * (\frac{N-i}{N})^2$
- Aditivo exponencial: $T_i = T_N + (T_0 - T_N) * \frac{1}{(1 + \exp(\frac{2 * \log(T_0 - T_N)}{N} * (i - \frac{1}{2} * N)))}$
- Aditivo trigonométrico: $T_i = T_N + \frac{1}{2}(T_0 - T_N) * (1 + \cos(\frac{i * \pi}{N}))$

- Multiplicativos

- Multiplicativo linear: $T_i = \frac{T_0}{1 + \alpha * i}$ com $\alpha > 0$
- Multiplicativo quadrático: $T_i = \frac{T_0}{1 + \alpha * i^2}$ com $\alpha > 0$
- Multiplicativo exponencial: $T_i = T_0 * \alpha^i$ com $0.8 \leq \alpha \leq 0.9$
- Multiplicativo logarítmico: $T_i = \frac{T_0}{1 + \alpha * \log(1 + i)}$ com $\alpha > 1$

- A busca tabu, do inglês *tabu search* (TS), foi formalizado em 1986 por Glover [Glover 1986], mesmo trabalho que introduziu o termo "meta-heurística".
- Usa explicitamente o histórico da busca como uma memória adaptativa, possibilitando escapar de mínimos locais e implementar uma estratégia de exploração.
- O uso dessa memória das soluções já encontradas no espaço de busca, torna a busca mais **econômica** e **efetiva**.
- o SA faz o caminho oposto, que não usa memória e, dessa forma, é incapaz de aprender com o passado e pode perder uma solução melhor que foi encontrada durante o processo de busca.

- O algoritmo inicia com uma solução $s = s_0$ e, a cada iteração, explora um subconjunto de soluções V na vizinhança $N(s)$ da solução atual s .
- A melhor solução encontrada nesse subconjunto s' se torna a solução atual s , mesmo que possua um valor $f(s') > f(s)$, considerando um problema de minimização.
- Essa estratégia de escolha do melhor vizinho visa escapar de mínimos locais; entretanto, isso pode fazer com que o algoritmo fique em um ciclo infinito, retornando à soluções visitadas anteriormente.
- Para evitar isso, a TS utiliza a **lista tabu** T , que dá nome ao algoritmo.

- Ela armazena as últimas soluções encontradas e, assim, impede que elas sejam revisitadas.
- Ela funciona como uma lista de tamanho fixo que armazena as últimas $|T|$ soluções visitadas, ou seja, a cada nova solução incluída na lista, a mais antiga é removida.
- Assim, na exploração do subconjunto V da vizinhança $N(s)$ da solução atual, a TS fica proibida de visitar as soluções s' já vistas anteriormente, enquanto permanecerem na lista tabu.
- Apesar das vantagens, a lista tabu também pode atrapalhar o processo de busca, proibindo movimentos de melhora da solução ou até mesmo levando a total estagnação da busca.

- Dessa forma, a inclusão de um critério de aspiração A , pode aprimorar bastante o processo de busca.
- Ele é um instrumento usado para passar pelas restrições impostas pela lista tabu, ou seja, retirar o status de tabu de uma certa solução s' , em certas circunstâncias.
- Há basicamente dois tipos de aspiração: por objetivo e a *default*.
 - Por objetivo: um movimento tabu para uma solução s' em V é permitido se $f(s') < A(f(s))$.
 - *Default*: o movimento tabu mais antigo é permitido se não houver qualquer movimento possível fora da lista tabu.

Busca Tabu

Input: Uma solução inicial s escolhida aleatoriamente

Input: Lista tabu de tamanho l vazia

Input: Número máximo de iterações $BTmax$

Input: Caso conheça o limite inferior, forneça f_{min}

$Iter = 0$

$MelhorIter = 0$

while $(Iter - MelhorIter) < BTmax$ **and** $f(s) > f_{min}$ **do**

$Iter = Iter + 1$

 Gere um subconjunto V na vizinhança $N(s)$

 Recupere a melhor solução s' de V tal que não seja tabu ou $f(s') < A(f(s))$

 Atualize a lista tabu; $s = s'$

if $f(s) < f(s^*)$ **then**

$s^* = s$

$MelhorIter = Iter$

end

end

Output: A melhor solução s^* encontrada

Variable Local Search

- O método de Busca em Vizinhança Variável (Variable Neighborhood Search, VNS) proposto por Hansen [Hansen e Ostermeier 2001].
- É uma metaheurística de busca local que consiste na exploração de uma vizinhança dinamicamente variável para uma dada solução.
- Na inicialização do algoritmo, as estruturas de vizinhança podem ser escolhidas arbitrariamente, mas normalmente como uma sequência de vizinhanças $N_1, N_2, \dots, N_{N_{max}}$ com cardinalidade crescente.
- Para que o VNS se torne eficiente, o ideal é que as estruturas de vizinhança usadas sejam complementares, impedindo que várias soluções sejam revisitadas.

Variable Local Search

- Em sua versão original, o VNS faz uso do método de Descida em Vizinhança Variável (Variable Neighborhood Descent, VND) [Hansen e Ostermeier 2001] para realizar a busca local.
- O algoritmo começa com uma solução inicial arbitrária e a cada iteração uma solução s' é escolhida aleatoriamente dentro da vizinhança $N_n(s)$ da solução atual s .
- Então, s' é usada como solução inicial de um método de busca local, produzindo s'' como solução ótima local.
- A busca local pode usar qualquer estrutura de vizinhança, não precisando se limitar à sequência definida para o VNS.

Variable Local Search

- Se s'' for melhor que s , s'' substitui s como solução atual e o ciclo recomeça com $n = 1$.
- Caso contrário, a busca continua na próxima vizinhança $n = n + 1$.
- Este procedimento termina quando um critério de parada é atendido.

Variable Local Search

Input: Selecione as estruturas de vizinhança (N_1, N_2, \dots, N_n) , onde $n = 1, 2, \dots, N_{max}$

Input: Escolha uma solução inicial s

while critério de parada não é atingido **do**

$n = 1$

while $n < N_{max}$ **do**

 Escolha aleatoriamente uma solução s' em $N_n(s)$

 Faça uma busca local começando em s' e obtenha s'' como ótimo local

if s'' é melhor que s **then**

$s = s''$

$n = 1$

end

else

$n = n + 1$

end

end

end

Output: A melhor solução s^* encontrada

Iterated Local Search

- A definição da metaheurística *Iterated Local Search* (ILS) foi dada por Lourenço e colaboradores em [Lourenço, Martin e Stützle 2002].
- Mas existem outros trabalhos que se enquadram como instâncias do ILS, como:
 - *Iterated descent* [Baum 1986];
 - *Large-step Markov chains* [Martin, Otto e Felten 1992];
 - *Iterated Lin-Kernighan* [Johnson 1990];
 - *Chained local optimization* [Martin e Otto 1993].

Iterated Local Search

- Segundo [Lourenço, Martin e Stützle 2002], há duas características principais que fazem de algoritmo instância do ILS.
 - ❶ Deve existir apenas uma trajetória sendo seguida, excluindo assim, os algoritmos baseados em populações;
 - ❷ A busca por melhores soluções deve acontecer em espaços de busca reduzidos que são definidos por uma heurística "caixa-preta".
- Na prática, essa heurística é implementada como uma busca local.

Iterated Local Search

- Ao invés de realizar buscas locais em soluções geradas aleatoriamente repetidamente, a ILS produz uma solução inicial para a próxima iteração através de uma **perturbação** do mínimo local encontrado na iteração atual.
- Esse procedimento é realizado na esperança que a perturbação gere uma solução que esteja localizada na bacia de atração de um mínimo local melhor.
- Esse mecanismo de perturbação é um ponto-chave da ILS
 - Se a perturbação for muito fraca, pode não ser suficiente para escapar da bacia de atração do mínimo local atual;
 - Se a perturbação for muito forte, pode fazer com que o algoritmo reinicie as buscas locais de maneira similar à aleatória.

Iterated Local Search

Input: Escolha de forma aleatória uma solução s

Realize uma busca local a partir de s , gerando s^*

while *critério de parada não é atingido* **do**

 Aplique uma perturbação sobre s^* para gerar uma solução p

 Faça uma busca local começando em p e obtenha p^* **while** $n < N_{max}$ **do**

 Escolha aleatoriamente uma solução s' em $N_n(s)$

 Faça uma busca local começando em s' e obtenha s'' como ótimo local

if *critério de aceitação é satisfeito* **then**

$s^* = p^*$

end

end

end

Output: A melhor solução s^* encontrada

- 1 Introdução
- 2 Metaheurísticas Baseadas em Solução Única
 - Simulated Annealing
 - Busca Tabu
 - Variable Local Search
 - Iterated Local Search
- 3 Metaheurísticas Baseadas em População**
 - Algoritmos Genéticos
- 4 Práticas

Metaheurísticas Baseadas em População

- Como o próprio nome sugere, trabalham com um conjunto de soluções ao invés de apenas uma.
- Os métodos mais estudados são relativos à **Computação Evolutiva (EC)** e **Inteligência de Partículas (SI)**.
- Os algoritmos evolutivos são inspirados na Teoria da Evolução de Darwin, onde os indivíduos de uma população evoluem através de recombinações e mutações.
- Já em SI, a ideia é produzir inteligência computacional através da exploração das interações sociais.
- Aqui vamos focar em **Algoritmos Genéticos**, que são uma vertente dos algoritmos evolutivos.

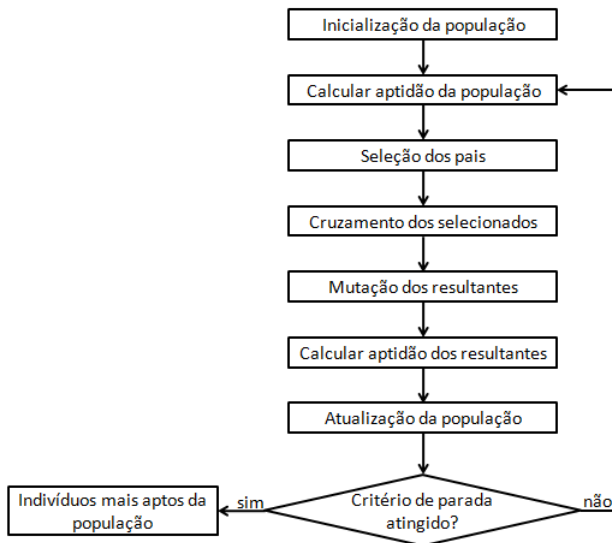
- O Algoritmo Genético (GA) [Goldberg 1989] é com certeza o método de computação evolutiva mais amplamente conhecido e utilizado.
- Foi desenvolvido originalmente na década de 70 por John Holland e seus alunos, cujos interesses eram direcionados ao estudo de sistemas adaptativos.
- Cada solução em potencial para o problema a ser resolvido se encontra codificada em uma estrutura chamada de **cromossomo** ou indivíduo, e o conjunto de soluções é chamado de **população**.
- Geralmente, a população inicial é gerada aleatoriamente e será modificada seguindo os processos biológicos da evolução através de gerações.

- A cada geração, novos indivíduos são gerados através de operadores genéticos e a população resultante será constituída por indivíduos considerados melhores do que aqueles presentes na geração passada.
- Os melhores indivíduos são aqueles que melhor se ajustam ao problema.
- A adequação de cada indivíduo é medida por uma **função de aptidão**, que indica quão boa é a solução para o problema em mãos.
- Os indivíduos mais aptos são mais prováveis de serem escolhidos e eles passam através das gerações até que alcançam a população final, de onde a melhor solução para o problema é encontrada.

- Dependendo da técnica de seleção utilizada, os melhores indivíduos podem ser movidos diretamente para a próxima geração ou selecionados para produzir descendentes pela aplicação de operadores genéticos tais como **crossover** e **mutação**.
- Tanto a função de aptidão quanto a forma de codificar as soluções ficam **dependendo** do domínio do problema.

- O fluxograma do funcionamento de um AG clássico é o seguinte:
 - 1 O processo de busca se inicia com uma população inicial gerada de alguma maneira, por exemplo, de forma aleatória.
 - 2 Os valores de aptidão de cada indivíduo da população são calculados.
 - 3 O método de seleção extrai pares de cromossomos da população atual para gerar novos indivíduos.
 - 4 O operador de *crossover* realiza a troca de segmentos entre dois cromossomos com certa probabilidade.
 - 5 Então, o operador de mutação é aplicado nos novos indivíduos para provocar mudanças em diferentes partes do cromossomo com certa probabilidade.
 - 6 Os indivíduos mais aptos são mantidos na próxima geração.
 - 7 O processo continua (passos 2-6) até que uma condição de parada seja satisfeita.
 - 8 Por fim, a melhor solução obtida através do processo é recuperada.






Algoritmos Genéticos











- 1 Introdução
- 2 Metaheurísticas Baseadas em Solução Única
 - Simulated Annealing
 - Busca Tabu
 - Variable Local Search
 - Iterated Local Search
- 3 Metaheurísticas Baseadas em População
 - Algoritmos Genéticos
- 4 Práticas

Hora das Práticas!

- ❶ Implementação de uma metaheurística baseado em solução única.
 - Testar em alguma função matemática e verificar a solução encontrada como sendo o valor **mínimo** dessa função.
- ❷ Implementação de uma metaheurística baseado em população.
 - Testar em alguma função matemática e verificar a solução encontrada como sendo o valor **mínimo** dessa função.

-  BAUM, E. B. Towards practical 'neural' computation for combinatorial optimization problems. In: *AIP Conference Proceedings*. AIP, 1986. v. 151, p. 53–58. ISSN 0094243X. Disponível em: <<http://aip.scitation.org/doi/abs/10.1063/1.36219>>.
-  BIRATTARI, M. et al. Classification of metaheuristics and design of experiments for the analysis of components. In: *AIDA-01-05*. [S.l.: s.n.], 2001. p. 1–12.
-  CERNY, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, v. 45, n. 1, p. 41–51, 1985. ISSN 1573-2878.
-  GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, Elsevier Science Ltd., Oxford, UK, UK, v. 13, n. 5, p. 533–549, maio 1986. ISSN 0305-0548.
-  GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. [S.l.]: Addison-Wesley, 1989. 432 p. (Artificial Intelligence). ISBN 0201157675.

-  HANSEN, N.; OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 9, n. 2, p. 159–195, jun. 2001. ISSN 1063-6560.
-  JOHNSON, D. S. Local optimization and the traveling salesman problem. In: *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*. London, UK, UK: Springer-Verlag, 1990. (ICALP '90), p. 446–461. ISBN 3-540-52826-1. Disponível em: <<http://dl.acm.org/citation.cfm?id=646244.684359>>.
-  KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. ISSN 00368075.
-  LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: *Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*. [S.l.]: Kluwer Academic Publishers, 2002. p. 321–353.

-  MARTIN, O.; OTTO, S. W.; FELTEN, E. W. Large-step markov chains for the tsp incorporating local search heuristics. *Oper. Res. Lett.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 11, n. 4, p. 219–224, maio 1992. ISSN 0167-6377. Disponível em: <[http://dx.doi.org/10.1016/0167-6377\(92\)90028-2](http://dx.doi.org/10.1016/0167-6377(92)90028-2)>.
-  MARTIN, O. C.; OTTO, S. W. *Combining Simulated Annealing with Local Search Heuristics*. [S.l.], 1993.
-  METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, v. 21, p. 1087–1092, 1953.
-  SIVANANDAM, S. N.; DEEPA, S. N. *Introduction to Genetic Algorithms*. [S.l.: s.n.], 2008. v. 2. 442 p. ISBN 9783540731894.