* [CASA](http://www.theprojectspot.com/home)
* [SOBRE MIM](http://www.theprojectspot.com/about-me)
* [TUTORIAIS](http://www.theprojectspot.com/tutorials)
* [BLOG](http://www.theprojectspot.com/blog)
* [NOTÍCIA](http://www.theprojectspot.com/news)

11 DE ABRIL DE 2013 · POR [LEE JACOBSON](http://www.theprojectspot.com/about-me)

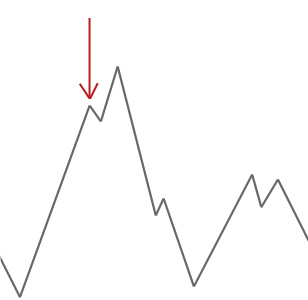
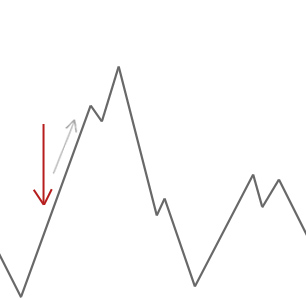
[Recozimento Simulado para iniciantes](http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6)

Encontrar uma solução ideal para determinados problemas de otimização pode ser uma tarefa incrivelmente difícil, muitas vezes praticamente impossível. Isso ocorre porque quando um problema é suficientemente grande, precisamos procurar por uma enorme quantidade de soluções possíveis para encontrar o melhor. Mesmo com o poder de computação moderno ainda existem muitas outras soluções possíveis a serem consideradas. Neste caso, porque não podemos realisticamente esperar encontrar o melhor dentro de um período de tempo sensível, temos que nos contentar com algo suficientemente próximo.  
  
Um problema de otimização de exemplo que geralmente tem um grande número de soluções possíveis seria o problema do vendedor ambulante. Para encontrar uma solução para um problema, como o problema dos vendedores ambulantes, precisamos usar um algoritmo capaz de encontrar uma solução suficientemente boa em um período razoável de tempo. Em um tutorial anterior, analisamos como podemos fazer isso com algoritmos genéticos e, embora os algoritmos genéticos sejam uma maneira de encontrar uma solução "boa o suficiente" para o problema do vendedor ambulante, existem outros algoritmos mais simples que podemos implementar, que também Encontre-nos uma solução próxima à ótima. Neste tutorial, o algoritmo que usaremos é "recozimento simulado".   
  
Se você não está familiarizado com o problema do vendedor ambulante, pode valer a pena ver o meu [tutorial anterior](http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5) Antes de continuar.

O que é Recozimento Simulado?

Primeiro, vejamos como o recozimento simulado funciona e por que é bom encontrar soluções para o problema dos vendedores ambulantes em particular. O algoritmo de recozimento simulado foi originalmente inspirado no processo de recozimento no trabalho de metal. O recozimento envolve o aquecimento e o resfriamento de um material para alterar suas propriedades físicas devido às mudanças na estrutura interna. À medida que o metal esfria, sua nova estrutura se torna fixa, fazendo com que o metal mantenha suas propriedades recém-obtidas. No recozimento simulado, mantemos uma variável de temperatura para simular esse processo de aquecimento. Nós inicialmente configurá-lo alto e, em seguida, permitir que ele lentamente "legal" como o algoritmo é executado. Embora esta variável de temperatura seja alta, o algoritmo será permitido, com mais frequência, para aceitar soluções que são piores do que a nossa solução atual. Isso dá ao algoritmo a capacidade de saltar de qualquer otimização local que se encontra no início da execução. À medida que a temperatura é reduzida, a possibilidade de aceitar soluções pior, permitindo assim que o algoritmo se centre gradualmente em uma área do espaço de busca em que, com sorte, uma solução próxima da ótima solução. Esse processo gradual de "resfriamento" é o que torna o algoritmo de recozimento simulado extraordinariamente eficaz para encontrar uma solução próxima à ótima quando lida com grandes problemas que contêm inúmeros otimizações locais. A natureza do problema dos vendedores ambulantes faz dele um exemplo perfeito. Permitindo que o algoritmo se concentre gradualmente em uma área do espaço de busca em que, com sorte, uma solução próxima da ótima solução. Esse processo gradual de "resfriamento" é o que torna o algoritmo de recozimento simulado extraordinariamente eficaz para encontrar uma solução próxima à ótima quando lida com grandes problemas que contêm inúmeros otimizações locais. A natureza do problema dos vendedores ambulantes faz dele um exemplo perfeito. Permitindo que o algoritmo se concentre gradualmente em uma área do espaço de busca em que, com sorte, uma solução próxima da ótima solução. Esse processo gradual de "resfriamento" é o que torna o algoritmo de recozimento simulado extraordinariamente eficaz para encontrar uma solução próxima à ótima quando lida com grandes problemas que contêm inúmeros otimizações locais. A natureza do problema dos vendedores ambulantes faz dele um exemplo perfeito.

Vantagens do recozimento simulado

Você pode estar se perguntando se há alguma vantagem real na implementação de recozimento simulado sobre algo como um alpinista simples. Embora os alpinistas possam ser surpreendentemente eficazes para encontrar uma boa solução, eles também têm uma tendência a ficar presos nos ótimos locais. Conforme determinado anteriormente, o algoritmo de recozimento simulado é excelente para evitar esse problema e é muito melhor, em média, em encontrar um ótimo ótimo global.   
  
Para ajudar a compreender melhor, vejamos rapidamente por que um algoritmo básico de escalada de colinas é tão propenso a ficar preso nos ótimos locais.   
  
Um algoritmo alpinista simplesmente aceitará soluções vizinhas que são melhores que a solução atual. Quando o montanhista não consegue encontrar melhores vizinhos, ele pára.  
  


No exemplo acima, começamos o nosso montanhista na seta vermelha e ele funciona até a colina até atingir um ponto em que não pode subir mais alto sem primeiro descer. Neste exemplo, podemos ver claramente que está preso em um ótimo local. Se este fosse um problema do mundo real, não saberíamos como o espaço de pesquisa parece tão desafortunado que não seríamos capazes de saber se esta solução está próxima de um ótimo global.   
  
O recozimento simulado funciona de forma ligeiramente diferente do que isso e, ocasionalmente, aceita piores soluções. Essa característica do recozimento simulado ajuda-o a saltar de qualquer otimização local que de outra maneira poderia ficar atrapalhada.

Função de aceitação

Vamos dar uma olhada em como o algoritmo decide quais as soluções a aceitar para que possamos entender melhor como é capaz de evitar esses otimizações locais.   
  
Primeiro, verificamos se a solução vizinha é melhor do que a nossa solução atual. Se for, aceitamos isso incondicionalmente. Se, no entanto, a solução vizinha não é melhor, precisamos considerar alguns fatores. Em primeiro lugar, quanto pior a solução do vizinho é; E em segundo lugar, quão alta é a atual "temperatura" do nosso sistema. Em altas temperaturas, o sistema é mais provável que aceite soluções que sejam pior.   
  
A matemática para isso é bastante simples:

**Exp ((soluçãoEnergia - vizinhançaEnergia) / temperatura)**

Basicamente, quanto menor a mudança de energia (a qualidade da solução), e quanto maior a temperatura, mais provável é que o algoritmo aceite a solução.

Visão geral do Algoritmo

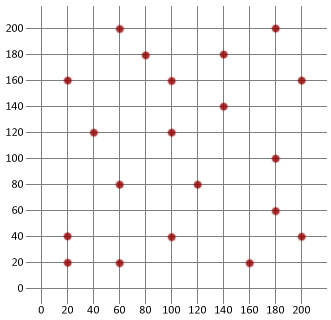
Então, como o algoritmo parece? Bem, em sua implementação mais básica, é bastante simples.

* Primeiro, precisamos definir a temperatura inicial e criar uma solução inicial aleatória.
* Em seguida, começamos a fazer um loop até que nossa condição de parada seja atendida. Normalmente, o sistema tem suficientemente resfriado, ou uma solução suficientemente boa foi encontrada.
* A partir daqui, selecionamos um vizinho fazendo uma pequena mudança na nossa solução atual.
* Então, decidimos se mudar para a solução do vizinho.
* Finalmente, diminuímos a temperatura e continuamos em loop

Inicialização da temperatura

Para uma melhor otimização, ao inicializar a variável de temperatura, devemos selecionar uma temperatura que inicialmente permita praticamente qualquer movimento em relação à solução atual. Isso dá ao algoritmo a capacidade de explorar melhor todo o espaço de busca antes de resfriar e se estabelecer em uma região mais focada.

Código de exemplo

Agora, vamos usar o que sabemos para criar um algoritmo básico de recozimento simulado e, em seguida, aplicá-lo ao problema vendedor vendedor abaixo. Nós vamos usar o Java neste tutorial, mas a lógica deve ser suficientemente simples para copiar para qualquer idioma de sua escolha.  
  


Primeiro, precisamos criar uma classe da cidade que possa ser usada para modelar os diferentes destinos do nosso vendedor ambulante.   
  
**City.java**

/ \*   
\* City.java   
\* Modelos de uma cidade   
\* /  **pacote sa; Classe   
  
pública City { int x; Int ;**// Constrói uma cidade **pública pública**aleatoriamente colocada**() { this .x = ( int ) (Math.random () \* 200); Isto .y = ( int ) (Math.random () \* 200);     }**// Constrói uma cidade em x escolhida, y localização **pública Cidade ( int x, int y) { this .x = x; Isto .y = y;     }**// Obtém a coordenada x da cidade **int público  
      
      
      
      
      
          
          
  
      
      
      
          
          
  
      
      
     GetX () {   
        retorna isso .x;   
    }**// Obtém a coordenada de cidade  **pública int getY () {   
        retorna este .y;   
    }**// Obtém a distância para a cidade  **pública pública pública de dupla distânciaTo (City city) {   
        int xDistance = Math.abs (getX () - city.getX ());   
        Int yDistance = Math.abs (getY () - city.getY ()); Distância   
        dupla = Math.sqrt ((xDistance \* xDistance) + (yDistance \* yDistance)); Distância de   
          
        retorno ;   
    }   
      
    @Override   
    public String toString () {   
        return getX () + "," + getY ();  
    }   
}**

Em seguida, crie uma aula que possa acompanhar as cidades.   
  
**TourManager.java**

/ \*   
\* TourManager.java   
\* Possui as cidades de um tour   
\* /  **pacote sa;   
  
Importar java.util.ArrayList; Classe   
  
pública TourManager {**// Contém nossas cidades **private static ArrayList destinationCities = nova ArrayList <City> ();**// Adiciona uma cidade de destino **public static void addCity (City city) {         destinationCities.add (city);     }**// Obter uma cidade **pública estática City getCity ( int index) { return (Cidade) destinationCities.get (índice);     }**// Obter o número de cidades de destino  **public static int numberOfCities () {   
        return destinationCities.size ();   
    }   
      
}**

Agora, crie a classe que pode modelar uma turnê de vendedor ambulante.   
  
**Tour.java**

/ \*   
\* Tour.java   
\* Armazena um passeio candidato por todas as cidades   
\* /  **pacote sa;   
  
Importar java.util.ArrayList;   
Importar java.util.Collections; Tour de classe   
  
pública {**// Realiza o nosso tour das cidades **Private ArrayList tour = new ArrayList <City> ();**// Cache **private int distance = 0;**// Constrói uma turnê **pública**tour em branco**() { for ( int i = 0; i <TourManager.numberOfCities (); i ++) {             tour.add (null);         }     }**// Constrói um tour de outra turnê  **pública Tour (ArrayList tour) {   
        this .tour = (ArrayList) tour.clone ();   
    }**// Retorna informações de turno  **public ArrayList getTour () { tour de   
        retorno ;   
    }**// Cria um **vazio   
    publicitário**individual aleatório**generateIndividual () {**// Bucle em todas as nossas cidades de destino e adicione-as ao nosso tour **para ( int cityIndex = 0; cityIndex <TourManager.numberOfCities (); cityIndex ++) {           setCity (cityIndex, TourManager .getCity (cityIndex));         }**// Reordenar aleatoriamente o tour**Collections.shuffle (tour);  
          
          
  
  
          
  
    }**// Obtém uma cidade da turnê  **pública City getCity ( int tourPosition) {   
        return (City) tour.get (tourPosition);   
    }**// Define uma cidade em uma determinada posição dentro de uma excursão  **pública vazio setCity ( int tourPosition, cidade Cidade) {   
        tour.set (tourPosition, cidade);**// Se as excursões foram alteradas, precisamos reiniciar a distância de aptidão e **distância = 0;   
    }**// Obtém a distância total do tour  **public int getDistance () {   
        if (distance == 0) {   
            int tourDistance = 0;**// Percorra as cidades do nosso passeio **Para ( int cityIndex = 0; cityIndex <tourSize (); cityIndex ++) {**// Obter cidade que estamos viajando de **City fromCity = getCity (cityIndex);**// Cidade, estamos viajando para **City destinationCity;**// Certifique-se de que não estamos na última cidade do nosso passeio , se formos a cidade de destino final da nossa turnê para a nossa cidade inicial  **se (cityIndex + 1 <tourSize ()) {   
                    destinationCity = getCity (cityIndex + 1);   
                }   
                Else {   
                    destinationCity = getCity (0);   
                }**// Obter a distância entre as duas cidades **tourDistance + = fromCity.distanceTo (destinationCity);   
            }   
            Distance = tourDistance;   
        } Distância de   
        retorno ;   
    }**// Obter número de cidades no nosso tour  **public int tourSize () {   
        return tour.size ();   
    }   
      
    @Override   
    public String toString () {   
        String geneString = "|" ;   
        Para ( int i = 0; i <tourSize (); i ++) {   
            geneString + = getCity (i) + "|" ;   
        }   
        RetornoGeneString;   
    }   
}**

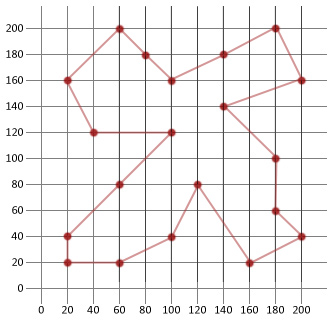
Finalmente, vamos criar nosso algoritmo de recozimento simulado.   
  
**SimulatedAnnealing.java**

**Pacote sa; Classe   
  
pública SimulatedAnnealing {**// Calcule a probabilidade de aceitação **public static double acceptanceProbability ( int energy, int newEnergy, double temperature) {**// Se a nova solução for melhor, aceite-a **se (newEnergy <energy) { return 1.0;         }**// Se a nova solução for pior, calcule um **retorno de**probabilidade de aceitação**Math.exp ((energy - newEnergy) / temperature);     } Público estático vazio  
  
      
       
          
          
              
  
          
          
  
  
     Principal (String [] args) {**// Crie e adicione nossas cidades **City city = new City (60, 200);   
        TourManager.addCity (cidade);   
        City city2 = new City (180, 200);   
        TourManager.addCity (city2);   
        Cidade city3 = cidade nova (80, 180);   
        TourManager.addCity (city3);   
        Cidade city4 = cidade nova (140, 180);   
        TourManager.addCity (city4);   
        City city5 = new City (20, 160);   
        TourManager.addCity (city5);   
        City city6 = new City (100, 160);   
        TourManager.addCity (city6);  
        Cidade city7 = cidade nova (200, 160);   
        TourManager.addCity (city7);   
        City city8 = new City (140, 140);   
        TourManager.addCity (city8);   
        City city9 = cidade nova (40, 120);   
        TourManager.addCity (city9);   
        Cidade city10 = cidade nova (100, 120);   
        TourManager.addCity (city10);   
        City city11 = cidade nova (180, 100);   
        TourManager.addCity (city11);   
        City city12 = cidade nova (60, 80);   
        TourManager.addCity (city12);   
        City city13 = cidade nova (120, 80);  
        TourManager.addCity (city13);   
        Cidade city14 = cidade nova (180, 60);   
        TourManager.addCity (city14);   
        City city15 = cidade nova (20, 40);   
        TourManager.addCity (city15);   
        City city16 = cidade nova (100, 40);   
        TourManager.addCity (city16);   
        City city17 = cidade nova (200, 40);   
        TourManager.addCity (city17);   
        City city18 = cidade nova (20, 20);   
        TourManager.addCity (city18);   
        City city19 = cidade nova (60, 20);   
        TourManager.addCity (city19);  
        City city20 = new City (160, 20);   
        TourManager.addCity (city20);**// conjunto inicial temporário  **duplo temp = 10000;**// Taxa de resfriamento  **duplo arrefecimento = 0,003;**// Inicializar a solução inicial **Tour currentSolution = new Tour ();   
        CurrentSolution.generateIndividual ();   
          
        System.out.println ( "Distância da solução inicial:" + currentSolution.getDistance ());**// Definir como melhor atual best best **tour = new Tour (currentSolution.getTour ());**// Loop até o sistema ter resfriado  **enquanto (temp> 1) {**// Criar novo **Tour**vizinho**Tour newSolution = new Tour (currentSolution.getTour ());**// Obter uma posição aleatória no tour  **int tourPos1 = ( int ) (newSolution.tourSize () \* Math.random ());   
            Int tourPos2 = ( int ) (newSolution.tourSize () \* Math.random ());**// Obter as cidades em posições selecionadas na turnê **City citySwap1 = newSolution.getCity (tourPos1);   
            City citySwap2 = newSolution.getCity (tourPos2);**// Troque-os **newSolution.setCity (tourPos2, citySwap1);   
            NewSolution.setCity (tourPos1, citySwap2);**// Obter energia de soluções  **intCurrentEnergy = currentSolution.getDistance ();   
            Int neighbourEnergy = newSolution.getDistance ();**// Decida se devemos aceitar o vizinho  **se (acceptProbability (currentEnergy, neighbourEnergy, temp)> Math.random ()) {   
                currentSolution = new Tour (newSolution.getTour ());   
            }**// Acompanhe a melhor solução encontrada  **se (currentSolution.getDistance () <best.getDistance ()) {   
                best = new Tour (currentSolution.getTour ());   
            }**// **Temperatura do**sistema frio**\* = 1-coolingRate;   
        }   
  
        System.out.println ("Distância da solução final:" + best.getDistance ());   
        System.out.println ( "Tour:" + melhor);   
    }   
}**

**Saída**

**Distância da solução inicial: 1966   
Distância da solução final: 911   
Tour: | 180, 200 | 200, 160 | 140, 140 | 180, 100 | 180, 60 | 200, 40 | 160, 20 | 120, 80 | 100, 40 | 60 20 | 20, 20 | 20, 40 | 60, 80 | 100, 120 | 40, 120 | 20, 160 | 60, 200 | 80, 180 | 100, 160 | 140, 180 |**

Conclusão

Neste exemplo, conseguimos mais da metade da distância da nossa rota inicial gerada aleatoriamente. Esperamos que mostre o quão útil este algoritmo relativamente simples é aplicado a certos tipos de problemas de otimização.  
  


Autor

[](http://www.theprojectspot.com/about-me)Olá, sou Lee.   
Sou um desenvolvedor do Reino Unido que adora tecnologia e negócios. Aqui você encontrará artigos e tutoriais sobre coisas que me interessam. Se você quer me contratar ou saber mais sobre mim, vá até a minha página [sobre mim](http://www.theprojectspot.com/about-me)

Links Sociais

Compartilhar[**15**](http://www.theprojectspot.com/)

Tag

[Algoritmo](http://www.theprojectspot.com/tag/algorithm)[tsp](http://www.theprojectspot.com/tag/tsp)[simulado-recozimento](http://www.theprojectspot.com/tag/simulated-annealing)[java](http://www.theprojectspot.com/tag/java)[artificial-intelligence](http://www.theprojectspot.com/tag/artificial-intelligence)

Artigos relacionados

[Criando um algoritmo genético para iniciantes](http://www.theprojectspot.com/tutorial_post/creating-a-genetic-algorithm-for-beginners/3)

[O membro Bionic](http://www.theprojectspot.com/post/the-bionic-limb/22)

[Introdução às Redes Neurais Artificiais - Parte 1](http://www.theprojectspot.com/tutorial_post/introduction-to-artificial-neural-networks-part-1/7)

[Aplicando um algoritmo genético ao problema dos vendedores ambulantes](http://www.theprojectspot.com/tutorial_post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5)

[Automação no local de trabalho](http://www.theprojectspot.com/post/workplace-automation/24)

Comentários

Feed do Twitter

"@kepsico Devo esperar que não tenha uma líder feminina porque ela não pode representar ou entender meus problemas? Meu Deus. Eu sou ... https://t.co/JjMQAcTPfx"   
30 de agosto de 2017, 14:54:40 | [Ligação](https://twitter.com/leejacobson_/status/902998023374077957)

"@kepsico Uma pessoa trans também seria um problema para ela porque ela não é trans? E se não, por que a mesma lógica não se aplicaria aqui?"   
30 de agosto de 2017, 14:35:36 | [Ligação](https://twitter.com/leejacobson_/status/902993225866203136)

"@kepsico De que maneira relevante? Por que um homem com as mesmas ambições políticas não seria suficientemente bom?"   
30 de agosto de 2017, 14:34:20 | [Ligação](https://twitter.com/leejacobson_/status/902992905333350400)

"@kepsico Será que" reverter "não implica uma regressão para você? E se essa implicação não foi intencional, você concordaria, pelo menos, é enganador?"   
30 de agosto de 2017, 14:20:59 | [Ligação](https://twitter.com/leejacobson_/status/902989548795416578)

"Não importa se você é o mais qualificado, o que importa é que não" voltei para um homem "."   
30 de agosto de 2017, 13:49:01 | [Ligação](https://twitter.com/leejacobson_/status/902981501855440896)

"Igualdade de gênero do século 21.   
Https://t.co/H3lie1osS9"   
30 de agosto de 2017, 13:48:18 | [Ligação](https://twitter.com/leejacobson_/status/902981322934714369)

[@leejacobson\_](http://twitter.com/leejacobson_)

Tags populares

[Entrevistas de](http://www.theprojectspot.com/tag/interviews)[nó-js](http://www.theprojectspot.com/tag/node-js)[ACO](http://www.theprojectspot.com/tag/ACO)[redes neurais](http://www.theprojectspot.com/tag/neural-networks)[espaço](http://www.theprojectspot.com/tag/space)[redes neurais-](http://www.theprojectspot.com/tag/neural-networks)[artificiais](http://www.theprojectspot.com/tag/artificial-neural-networks)[simulados-recozimento](http://www.theprojectspot.com/tag/simulated-annealing)[perceptron-aprendizagem-regra](http://www.theprojectspot.com/tag/perceptron-learning-rule)[multicamadas-perceptron](http://www.theprojectspot.com/tag/multilayer-perceptron)[bionics](http://www.theprojectspot.com/tag/bionics)[startups](http://www.theprojectspot.com/tag/startups)[artificial-inteligência](http://www.theprojectspot.com/tag/artificial-intelligence)[algoritmos genéticos](http://www.theprojectspot.com/tag/genetic-algorithms)[ant-colonia-otimização](http://www.theprojectspot.com/tag/ant-colony-optimization)[hello-world](http://www.theprojectspot.com/tag/hello-world)[java](http://www.theprojectspot.com/tag/java)[algoritmo de](http://www.theprojectspot.com/tag/genetic-algorithm)[algoritmo](http://www.theprojectspot.com/tag/algorithm)[genético](http://www.theprojectspot.com/tag/genetic-algorithm)

Contrate-me

Estou disponível para trabalho freelancer,   
lee @ cwpstudios [dot] com

**Obrigado pela visita!**  
  
Mantenha contato,   
[envie-me um e-mail](mailto:lee@cwpstudios.com) ou [siga-me no Google+](https://plus.google.com/102308414709447205951?rel=author)

[Início](http://www.theprojectspot.com/home)| [Sobre Mim](http://www.theprojectspot.com/about-me)| [Projetos](http://www.theprojectspot.com/projects)| [Artigos](http://www.theprojectspot.com/articles)| [Tutoriais](http://www.theprojectspot.com/tutorials)| [Blog](http://www.theprojectspot.com/blog)  
  
Construído por [cwpStudios.com](http://www.cwpstudios.com/)  
Mantido e atualizado por [Lee Jacobson](http://www.theprojectspot.com/about-me)  
  
Copyright © 2017 The Project Spot. Todos os direitos reservados.