AT3: Têmpera Simulada e Algoritmos Genéticos

QXD0037 - Inteligência Artificial
Período: 2019.2

Samy Sá

Universidade Federal do Ceará
Campus de Quixadá
Quixadá, Brasil
samy@ufc.br

Requisitos: Conhecimentos de Programação, Representação de Problemas como Busca, Subida de Encosta e Busca pelo Melhor Primeiro.
Publicação: 08/11/2019
Prazo: 24/11/2019

1 Introdução

Este documento descreve o terceiro trabalho de implementação da disciplina e consiste em uma aplicação das técnicas de Têmpera Simulada (Simulated Annealing) e Algoritmos Genéticos (Genetic Algorithms). Com as técnicas do segundo trabalho, estas duas apresentam bons resultados em buscas locais com heurísticas admissíveis.

Trabalharemos o mesmo problema que no trabalho de implementação anterior, a dizer, o problema de alocação de enfermeiros em turnos de um hospital. Forneceremos novamente a representação do problema, e discussão sobre a avaliação dos estados. Neste problema cada solução é simplesmete um estado, como acontece no problema das n rainhas. Isto simplificará o trabalho de implementação, pois dispensa a manutenção de caminhos parciais nas estruturas de dados auxiliares.

2 Enunciado do Problema

O objetivo deste trabalho é atacar o problema de alocação de enfermeiros em turnos de trabalho em um hospital. Note que este problema é análogo a uma variedade de outros problemas de alocação de recursos. Para a representação dos estados do espaço de busca, assuma que há k enfermeiros denominados $e_1, e_2, \ldots, e_k$, a serem alocados em n turnos $t_1, t_2, \ldots, t_n$, e um conjunto de m restrições $r_1, r_2, \ldots, r_m$ que desejamos satisfazer em uma alocação adequada.

- As quantidades k de enfermeiros e n de turnos influenciam na representação dos estados: os estados do espaço de busca serão opções de alocação dos k enfeiros nos n turnos, o que pode ser pensado como uma matriz $k \times n$ ou string de tamanho $k.n$.
- As m restrições serão utilizadas para produzir a função de avaliação: dadas duas opções de alocações, a melhor das duas será a que mais se aproximar de satisfazer as restrições.

Com isto, todos os estados do espaço de busca terão k enfermeiros alocados, mas não necessariamente satisfarão às restrições fornecidas. Um estado será considerado objetivo se satisfizer a todas as restrições. O objetivo das nossas buscas será, portanto, encontrar uma opção de alocação dos enfermeiros que satisfaça a todas as restrições; caso seja impossível, desejamos minizar as violações a estas restrições. Qualquer estado poderá ser considerado inicial.

2.1 Representação de Estados

Os estados do problema podem ser facilmente visualizados na forma de matriz em que colocamos os enfermeiros nas linhas e os turnos nas colunas. Neste caso, cada célula i, j na matriz conterá o número 1, caso o enfermeiro i esteja alocado no turno j, ou o número 0, indicando o contrário. Por exemplo, em um problema com 5 enfermeiros e 3 turnos, a seguinte matriz representa um dos possíveis estados no espaço de busca:

	t_1	t_2	t_3
e_1	1	0	1
e_2	1	0	0
e_3	0	1	0
e_4	1	1	1
e_5	0	1	0

Este estado (chamemos-lhe de s_1) sugere alocarmos o enfermeiro e_1 nos turnos t_1 e t_3, enquanto e_2 deve ser alocado somente ao turno t_1, o enfermeiro e_3 somente ao turno t_2, o enfermeiro e_4 a todos os turnos, e o enfermeiro e_5 somente ao turno t_2. Neste exemplo, falamos em 3 turnos, mas note que poderíamos querer montar a escala dos enfermeiros para uma semana inteira em turnos de 8h (7h-15h; 15h-23h; 23h-07h), o que nos daria 21 turnos diferentes.

Cada estado também pode ser representado em forma de string, o que facilitará a implementação de algumas operações e também um dos nossos trabalhos futuros usando o mesmo problema. Por exemplo, o estado s_1 representado acima na matriz seria representado pela string "101100010111010".

2.2 Restrições e Função de Avaliação

O estado s_1 exibido acima nos é interessante, pois aparentemente o enfermeiro e_4 está sobrecarregado em comparação aos colegas e_3 e e_5. Neste tipo de problema, é possível que desejemos distribuir os turnos de trabalhos de forma mais egualitária, exemplificando uma das formas como um estado pode violar as restrições desejadas.

Suponha, por exemplo, que a única restrição do problema seja r_1 : "cada enfermeiro deve trabalhar somente 1 turno". Neste caso, o estado s_1 do nosso exemplo (codificado como "101100010111010") causa 2 violações, pois e_1, e_4 estariam sendo alocados em múltiplos turnos. A função de avaliação $g(x)$ deve, neste caso, retornar $g(s_1) = 2$, enquanto que cada estado de objetivo retornará 0 (zero). Em comparação, o estado s_2 : "100010101001010" causa apenas 1 violação, pois somente o enfermeiro e_3 trabalharia

mais que um turno; logo, $g(s_2) = 1$. Como obtemos $g(s_2) < g(s_1)$, o estado s_2 seria considerado melhor (mais próximo de um objetivo) que s_1.

Considere ainda o estado s_3 : "101100010101010", muito parecido com o estado s_1 e note que se nós contarmos violações à restrição r_1 como fizemos acima, teremos $g(s_3) = g(s_1)$. Podemos argumentar que s_3 deveria ser um pouco melhor que s_1, pois o enfermeiro e_4 trabalharia apenas 2 turnos em s_3, enquanto trabalharia os 3 turnos em s_1. Podemos corrigir isto retornando as violações a r_1 pela soma de quantos turnos a mais (ou a menos) que o devido cada enfermeiro trabalha. Neste caso, teríamos $g'(s_1) = 1+0+0+2+0 = 3$, $g'(s_2) = 0+0+1+0+0 = 1$ e $g'(s_3) = 1+0+0+1+0 = 2$, nos dando $g'(s_3) < g'(s_1)$.

Caso hajam múltiplas restrições no problema, a função de avaliação irá simplesmente somar as violações retornadas com base em cada uma destas restrições.

2.3 Vizinhanças no Espaço de Busca

Esta é a parte mais simples na representação do nosso problema para as técnicas de gerar e testar. Dado estado s qualquer, seus vizinhos serão obtidos variando seus bits um por vez. Os estados que representamos acima, por exemplo, terão 15 vizinhos cada. Para um exemplo mais simples de vizinhança, caso a string 01101 fosse um estado do problema, seus vizinhos seriam $11101, 00101, 01001, 01111, 01100$. Em cada caso, alteramos apenas um bit. A depender do estado que operarmos e das restrições do problema, teremos alguns vizinhos com avaliações melhores ou piores e as técnicas que utilizaremos se utilizarão disso. Por simplicidade, gere os vizinhos variando os bits na ordem dos seus índices na string.

3 Sugestões de Implementação

O seu primeiro trabalho é implementar a representação do problema, ou seja, uma forma de ler entrada e saída, um gerador de estados e o critério de parada. Esta parte é muito similar aos passos realizados no trabalho de Busca em Profundidade e Busca em Largura, mudando apenas o problema que abordaremos.

Comece com uma forma simples de instanciar estados como strings de bits, pois o gerador de estados será consideravelmente mais simples se utilizarmos a representação dos estados como strings. Por simplicidade, considere todas as execuções para a instância de 10 enfermeiros distribuídos em 21 turnos, de forma que as strings geradas terão tamanho fixo de 210 bits. Em geral, para as técnicas escolhidas, você não precisará se preocupar em controlar redundâncias neste gerador de estados.

Em seguida, implemente a função de avaliação dos estados. Isto promoverá uma pequena diferença em relação ao trabalho anterior, pois os estados serão inseridos na estrutura de dados auxiliar das buscas (uma fila) junto com o seu valor na função de avaliação. Lembre-se que as técnicas de Subida de Encosta (pelo Maior Aclive ou não) e Busca pelo Melhor Primeiro todas mantém os estados não visitados ordenados[1] pelos valores retornados na função de avaliação.

[1] Apenas parcialmente, no caso da Subida de Encosta.

Uma diferença em relação a outros problemas que discutimos, é que as técnicas atuais não podem garantir encontrar um estado de objetivo. Invés disso, lembre-se, elas tentam otimizar os valores de uma função de avaliação. No nosso caso, objetivaremos minimizar o número de violações às restrições do problema. O critério de parada, portanto, ocorrerá quando não tivermos nenhum estado gerado (a ser visitado) com avaliação melhor que o estado corrente (o que foi visitado mais recentemente). Note que se um estado objetivo for visitado, os algoritmos vão parar, pois estes terão avaliação 0, o que garantiria a minimização.

4 Enunciado Principal

Esta seção apresenta tudo o que você precisa implementar e discute alguns testes importantes.

4.1 Instância do Problema

Você deve utilizar seu código e compor uma função de avaliação para a alocação de 10 enfermeiros $e_1, e_2, \ldots, e_{10}$ de um hospital no turnos de uma semana. O hospital tem 3 turnos de 8h de trabalho por dia, todos os dias da semana, totalizando 21 turnos $t_1, t_2, \ldots, t_{21}$ para alocação de pessoal. Deve-se buscar satisfazer as seguintes restrições:

- r_1: Deve haver ao menos 1 enfermeiro e no máximo 3 enfermeiros em cada turno.
- r_2: Cada enfermeiro deve ser alocado em 5 turnos por semana.
- r_3: Nenhum enfermeiro pode trabalhar mais que 3 turnos seguidos sem folga.
- r_4: Enfermeiros preferem consistência em seus horários, ou seja, eles preferem trabalhar todos os dias da semana no mesmo turno (dia, noite, ou madrugada).

Cabe a você decidir como quantificar as violações com base em cada restrição. Recomenda-se, portanto, implementar uma função diferente para calcular as violações de cada restrição e obter como função de avaliação (a que desejamos minimizar) por alguma combinação destes valores. Isto permitirá que você revise a sua interpretação de cada restrição independentemente das demais, se necessário. Isso também permitirá que você rode simulações independentes com conjuntos diferentes destas restrições como forma de teste, se o desejar. Uma vez que seu código esteja completo, recomenda-se experimentar variações (caso você perceba possibilidades) e ver como isso afeta sua performance e os melhores resultados encontrados.

4.2 Técnicas

As seguintes técnicas devem ser implementadas para este problema:

1. Têmpera Simulada
2. Algoritmo Genético

Estas técnicas não mantém histórico dos estados testados em iterações anteriores, dispensando a utilização de estruturas de dados auxiliares. Dependendo de como você implementar a escolha dos estados a serem visitados na Têmpera Simulada, porém, pode ser útil utilizar um vetor, mas há um caminho mais eficiente obtido com uma pequena alteração no gerador de estados do segundo trabalho. A título de exemplo, recuperaremos os estados fictícios da Seção 2.3. Suponha que o estado corrente em uma execução da Têmpera Simulada seria 01101. Na próxima iteração, o algoritmo escolherá um (e apenas um) de seus vizinhso aleatoriamente para comparar-lhes. Você pode (1) gerar todos os vizinhos, guardar num vetor e depois sortear o índice; ou (2) escolher aleatoriamente qual bit alterar e gerar apenas este vizinho. Em ambos os casos, convém calcular a avaliação de cada estado gerado imediatamente quando gerá-lo, jamais depois. A decisão de manter o estado 01101 ou trocá-lo pelo vizinho sorteado segue conforme rege a técnica.

4.3 Parâmetros

A implementação de ambas as técnicas deve permitir a escolha de alguns parâmetros.

Têmpera Simulada Utilize uma agenda de resfriamento linear, ou seja, uma em que a temperatura inicial é reduzida de uma diferença constante a cada iteração. Por simplificade, você pode considerar que a diferença de temperatura é sempre de uma unidade. Desta forma, o único parâmetro do algoritmo será a temperatura inicial T_0. Um segundo parâmetro é requerido, a dizer, para indicação de estado inicial. Como no Trabalho 2, o estado inicial deve poder ser escolhido pelo usuário (para testes) ou gerado aleatoriamente. Recomenda-se escolher valores default para os parâmetros: estado inicial aleatório, temperatura inicial de 350. Encoraja-se experimentar com diferentes valores de temperatura inicial e ajustar o default da temperatura inicial de acordo com seus testes. Para estes testes, recomenda-se variar o estado inicial da mesma forma que proposto no Trabalho 2 para Subida de Encosta.

Algoritmo Genético Você pode implementar a variação de algoritmo genético que desejar quanto à forma de mutação e tipo de elitismo. Seu código deve permitir escolhermos valores para (i) tamanho da população, (ii) quantidade de gerações (iterações) desejadas, (iii) probabilidade de mutação e (iv) porcentagem de elitismo (4 parâmetros). Mais uma vez, recomenda-se escolher valores default para os parâmetros: 40 para tamanho da população, 120 gerações, 5% de probabilidade de mutação, 25% de elitismo. Igualmente, encoraja-se experimentar diferentes valores de parâmetros e ajustar os valores default de acordo com seus testes. Diferentes formas de implementar mutação e elitismo podem interferir com os valores sugeridos.

4.4 Entrada e Saída

Como entrada, forneça um menu simples que permita ao usuário escolher a técnica desejada e em seguida tenha opções de entrar seguir com os parâmetros default ou dar novas entradas.

Para saída da Têmpera Simulada, exiba na tela todos os estados avaliados. Idealmente, liste (em ordem) a temperatura corrente da iteração, o estado gerado para comparação, sua energia, e se este é ou não escolhido como novo estado. Ao final, destaque o estado em que a busca se encerra.

Para saída do Algoritmo Genético, recomenda-se a conversão das strings de bits para strings do sistema octal ou hexadecimal (apenas para exibição). Isso reduzirá o tamanho das strings a $1/3$ ou $1/4$ de seu tamanho original, permitindo a impressão de cada estado em uma única linha do terminal. Isto, por sua vez, permitirá a impressão da população de forma mais amigável aos usuários. Exiba um indivíduo por linha seguido do seu fitness e numerando-os conforme os índices do vetor que guardará indivíduos na sua implementação. Utilize estes números para indicar de forma concisa os emparelhamentos de indivíduos antes de formar a próxima geração.

5 Entrega

A submissão do trabalho se dará via SIGAA no código de atividade AT3. Você deve fazer upload de um arquivo .zip contendo o seguinte:

1. Um arquivo nomeado 'equipe.txt' com matrícula e nomes dos integrantes da equipe;
2. Todos os códigos de implementação;
3. Um executável, script ou arquivo principal de código a ser interpretado;
4. Um arquivo 'help.txt' com instruções para executar seu código, incluindo detalhes sobre eventuais interpretadores e versões requeridos, requisitos de sistema operacional, e operações para passar parâmetros ou iniciar a busca. O ideal é que seu código possa ser executado sem esforço. Caso você implemente outros controles, como um comando para pausar a busca ou avançá-la um passo por vez (podem ser úteis para debugging), indique também neste aquivo como utilizá-los.
5. Um arquivo de texto

Este enunciado foi disponibilizado em 08/11/19 e a submissão do trabalho deve ser feita até o dia 24/11/19.

6 Avaliação

Esta seção indicará os critérios de avaliação do que foi pedido e dá algumas opções de incrementos que podem contabilizar pontos extras. A intenção dos créditos extras é que possam compensar pequenas falhas nos quesitos principais e a recomendação é de que sejam tentadas somente após cumprir os quesitos principais.

- Até 1,0 pts pela documentação dos códigos: comentários adequados, compreensíveis, frequentes;
  - Indique o que cada trecho se propõe a resolver;
  - Utilize comentários para detalhar a modelagem que você está implementando.
- Até 2,0 pts. se a saída dos algoritmos estiver exibida conforme solicitado e de forma compreensível;

- Até 3,0 pts. se a Têmpera Simulada estiver adequadamente implementada;
- Até 3,0 pts. se o Algoritmo Genético estiver adequadamente implementado;
- Até 1,0 pts por elegância dos códigos e soluções: código limpo e compreensível, eficiente, etc;

Créditos extras:

- Até 1.5 pts. Produza um texto curto (2-3 páginas) discutindo como este trabalho teria lhe ajudado na compreensão das técnicas trabalhadas e demais conteúdos da disciplina de IA. Discuta neste texto os testes que você fez com a sua implementação e o que vocês observou nas execuções, documentando casos interessantes de estados iniciais que você tentou e os seus resultados.
- Até 1.0 pts. Implemente uma versão iterativa da Têmpera Simulada em que a temperatura é reiniciada ao fim da agenda de resfriamento. Há muito que pode mudar nesta alternativa e algumas decisões de design deverão ser tomadas por vocês. Indique os detalhes da sua implementação no arquivo 'help.txt', incluindo suas decisões quanto a critérios de parada, condições para reiniciar a busca, etc.
- Até 2,0 pts. Generalize o seu código para permitir executarmos as buscas variando a quantidade de enfermeiros. A quantidade de turnos e as restrições sugeridas permanecerão todas iguais. P.S.: é fortemente recomendado resolver o trabalho com estados de tamanho fixo e tentar a generalização somente ao final.