UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIENCIA DA COMPUTACAO
ALGORITMOS E ESTRUTURAS DE DADOS III
PRIMEIRO SEMESTRE DE 2014

Trabalho Prático 3 - Alocação de Tarefas entre Agentes Robóticos usando Técnicas de Coloração de Grafos

1. Introdução

Com o avanço da tecnologia computacional, umas das áreas que está sendo cada vez mais visada é o campo da robótica e suas aplicações. Primeiramente, agentes robóticos eram utilizados somente em linhas de montagem. Entretanto, atualmente eles podem estar presentes em nossas casas, realizando desde tarefas simples, como aspirar o pó, até tarefas mais complexas, como auxiliar no cuidado a um idoso.

Podemos citar diversas aplicações para robôs, mas uma que vem se destacando é o uso de agentes robóticos para o patrulhamento e monitoramento de áreas privadas, com a intenção de detectar intrusos ou mesmo fenômenos inesperados, como um incêndio ou vazamento de gás. A tarefa de patrulhamento neste âmbito é descrita pelo uso de um grupo de agentes capazes de ocupar determinados pontos na área a ser observada e coletar informações.

Um dos problemas que é intrínseco ao uso de uma equipe de agentes robóticos é o da alocação de tarefas entre eles. Esta alocação deve ser feita de forma a garantir o uso eficiente dos agentes, além de garantir que a área a ser monitorada seja totalmente inspecionada.

2. Definição do Problema

Neste trabalho, um algoritmo de alocação de tarefas para uma equipe de agentes robóticos deve ser desenvolvido. Para tanto, o aluno deve considerar o conjunto $\mathcal{P} = \{p_1, p_2, ..., p_n\}$ formado por n pontos a serem visitados pelos agentes e o conjunto $\mathcal{R} = \{r_1, r_2, ..., r_k\}$ de k agentes. Cada agente r_i possui um sensor capaz de realizar a varredura circular (isto é, 360º) com raio S. A área definida por este raio determina a área de observação do agente. O problema da alocação de tarefas entre os agentes robóticos pode ser mapeado no problema de Coloração de Grafos conforme descrito a seguir.

Cada ponto $p \in \mathcal{P}$ representa uma tarefa que deve ser alocada entre os agentes. Cada ponto p é descrito por suas coordenadas (x, y) e está localizado dentro da área a ser monitorada, além de ser mapeado em um vértice do grafo a ser construído.

As arestas entre dois vértices serão definidas como segue. Primeiramente serão criados subconjuntos de pontos que devem ser visitados por agentes distintos. Para tal, deve-se executar os seguintes passos para cada ponto p no conjunto P (representado por um vértice no grafo), estes passos são ilustrados na Figura 1:

1. Verifique quais pontos não estão dentro da área de observação do sensor de p. Para realizar esta etapa, deve ser calculada a distância entre o ponto de origem e

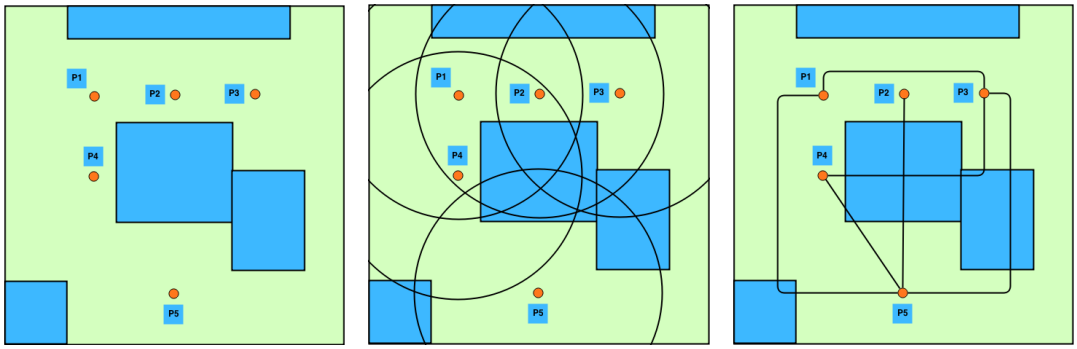
os demais pontos, aplicando a Equação da Distância Euclidiana (vide abaixo), se a distância for maior que o raio do S sensor, então o ponto está fora da área de atuação do sensor.
2. Para cada ponto q que está fora da área de atuação do sensor de p, adicione uma aresta (p,q) no grafo.

A Equação da Distância Euclidiana entre dois pontos p e q, definidos pelas coordenadas (x_p, y_p) e (x_q, y_q), respectivamente é dada por:

$$dist = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \tag{1}$$

Após a realização destas etapas, teremos um grafo conectado a ser utilizado para solução do problema de coloração. Cada agente no conjunto $\mathcal{R}$ representa uma cor que pode ser utilizada durante a resolução, porém, pretende-se utilizar a menor quantidade possível de agentes.

Para colorir o grafo, ou seja, distribuir as tarefas entre os agentes, a única regra a ser respeitada é a de que se dois pontos estiverem conectados, os mesmo não podem possuir a mesma cor, ou seja, não podem ser alocados para o mesmo agente. Ao fim da coloração, teremos uma distribuição de tarefas, como exemplo da Figura 2, onde foram utilizados 3 agentes.



(a) Distribuição dos pontos do conjunto $\mathcal{P}$ em suas respectivas localidades dentro da área a ser monitorada.

(b) Aplicação do passo 2 para criação do grafo, no qual são executados os testes de distância entre os pontos considerando o raio de ação do sensor.

(c) Aplicação do passo 3 para criação do grafo, onde são criadas conexões entre os pontos que estão fora do raio de atuação do sensor.

Figura 1. Etapas para a criação do grafo a ser utilizado para coloração.

2.1. Algoritmos

Com o intuito de demonstrar os conhecimentos adquiridos sobre paralelização, neste trabalho você **deve** apresentar a solução para a coloração de grafos com 3 (três) algoritmos distintos:

1. **Algoritmo exato de força bruta**: deve realizar a coloração de forma sequencial testando todas as possibilidades até achar uma válida.
2. **Algoritmo exato paralelo**: deve ser similar ao algoritmo de força bruta, porém realizar o procedimento de forma paralela.

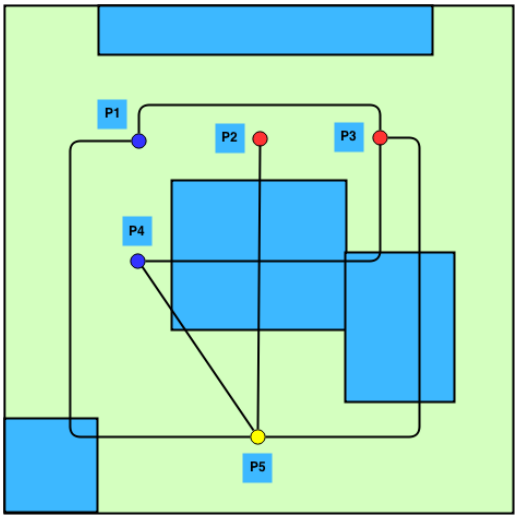


Figura 2. Distribuição dos pontos (tarefas) entre os agentes disponíveis. Foram utilizados 3 agentes.

3. **Algoritmo com Heurística**: implementa uma heurística arbitrária que direciona o algoritmo para a solução, o que deve ser bem diferente da força bruta.

Para seu algoritmo paralelo, é pedido que desenvolva uma estratégia de paralelização, esta pode compreender os assuntos que foram dados em sala de aula, bem como demais fontes de pesquisa. Esta estratégia deve ser detalhada em sua documentação, explanando os passos para sua execução e funcionamento.

3. Pontos Extras

Algoritmo Aproximado : algoritmo que apresenta uma solução aproximada da solução exata. Este algoritmo é *opcional* e será creditado valendo 4 (quatro) pontos extras durante a avaliação do trabalho. Você deverá descrevê-lo similarmente ao demais algoritmos, porém deve apresentar também sua **razão de aproximação**.

4. Entrada

Sua solução receberá como entrada um arquivo descrevendo um conjunto de estados. Cada estado possui um conjunto de pontos (tarefas) a serem visitados, a quantidade de robôs disponíveis e o raio do sensor utilizado pelos agentes.

O arquivo pode conter 1 (um) ou mais estados a serem estudados, cada estado estará separado dos demais por 1 (uma) linha em branco. A primeira linha de um estado descreve a quantidade de agentes disponíveis (valor inteiro) bem com o raio do sensor (valor de ponto flutuante), os valores estarão separados por 1 (um) espaço em branco. As demais linhas descrevem todos os pontos a serem visitados pelos robôs, cada ponto é descrito por um identificador (valor inteiro) e por suas coordenadas X e Y (podem ser de ponto flutuante). Estes dados estarão separados por 1 (um) espaço em branco.

A seguir temos um exemplo de arquivo de entrada:

```
3 5.0
1 -3 3
```

```
2 0 3
3 3 3
4 -3 0
5 0 -5

3 3.5
1 -4.5 0.5
2 0.5 2
3 1.3 -1
4 4.7 -3.7

2 3
1 -3 0
2 3 0
3 0 5
```

5. Saída

Sua solução deve criar na mesma pasta de execução um arquivo de saída chamado *coloring.txt* que descreverá a solução para a coloração do grafo que foi gerada pelos seus algoritmos *para cada estado*. O conteúdo deste arquivo deve seguir as seguintes regras:

- Cada linha deve listar todos aqueles pontos que devem ser alocados para o mesmo robô, ou seja, aqueles que foram coloridos com a mesma cor. Para tal, liste os identificadores dos pontos separados por 1 (um) espaço em branco, se houver mais de 1 (um) ponto.
- A solução para cada um dos estados processados deve ser separada por 1 (uma) linha em branco.
- Se não for possivel colorir o grafo com a quantidade de cores disponíveis, o estado não possui solução válida, então deve ser impresso o texto *'no solution'* no arquivo de saída.

A ordem em que os conjunto de pontos são listados fica a cargo de seu algoritmo, não é definida uma ordem para tal. Um exemplo de arquivo de saída é descrito a seguir:

```
1 4
2 3
5

1
2 3
4

no solution
```

6. Execução

O programa gerado deve ter o nome **tp3** e aceitar os seguintes argumentos:

```
./tp4 <algoritmo> <input>
```

algoritmo : um número do conjunto {1,2,3,4[1]}, onde cada elemento representa o algoritmo a ser executado respeitando a ordem definida na seção 2.1.

input : nome do arquivo que deve ser tratado como a entrada para o algoritmo.

7. Entrega

- A data **final** de entrega deste trabalho será 24/11/2014. Está data **não será prorrogada**.
- Submeta apenas um arquivo chamado <numero_matricula>_<nome>.zip. Não utilize espaços no nome do arquivo. Ao invés disso utilize o caractere '_'.
- Não inclua arquivos compilados ou gerados por IDEs. **Apenas** os arquivos abaixo devem estar presentes no arquivo zip:
  - Makefile
  - Arquivos fonte (*.c e *.h)
  - Documentação.pdf
- Não inclua **nenhuma pasta**. Coloque todos os arquivos na raiz do zip.
- Siga rigorosamente o formato do arquivo de saída descrito na especificação. Tome cuidado com *whitespaces* e formatação dos dados de saída.
- Não será necessário entregar a documentação impressa.
- Será adotada **médio harmônica** entre as notas da **documentação e da execução**, o que implica que a nota final será 0 se uma das partes não for apresentada.

8. Documentação

A documentação não deve exceder 10 páginas e deve conte pelo menos os seguintes itens:

- Uma **introdução** do problema em questão.
- **Modelagem e solução proposta** para o problema. O algoritmo deve ser explicado de forma clara, possivelmente através de pseudo-código e esquemas ilustrativos.
- Para cada algoritmo, você deve apresentar uma descrição do seu funcionamento e bem como um detalhamento de cada parte que achar importante.
- **Análise de complexidade** de tempo e espaço da solução implementada.
- **Experimentos** variando-se o tamanho da entrada e quaisquer outros parâmetros que afetem significativamente a execução.
  - Espera-se a utilização de tabelas e gráficos, com suas respectivas análises.
  - Na análise do algoritmo paralelo espera-se gráficos que apresentem o desempenho em função do número de threads, com pelo menos um gráfico mostrando a variação do tempo de execução em função do número de threads, por exemplo.
- Especificação da(s) **máquina(s) utilizada(s)** nos experimentos realizados.
- Uma breve **conclusão** do trabalho implementado.

[1] Sua solução deve receber este valor e executar o algoritmo extra, se o mesmo não foi implementado, o arquivo de saída *coloring.txt* deve conter o texto *'not implemented'*.

9. Código

- O código deve ser obrigatoriamente escrito na **linguagem C**. Ele deve compilar e executar corretamente nas máquinas Linux dos laboratórios de graduação.
- O utilitário **make** deve ser utilizado para auxiliar a compilação, um arquivo chamado *Makefile* deve gerar um executável que deverá obrigatoriamente ser capaz de receber o nome dos arquivos de entrada e saída com foi descrito na seção 6.
- As estruturas de dados devem ser **alocadas dinamicamente** e o código deve ser **modularizado** (divisão em múltiplos arquivos fonte e uso de arquivos cabeçalho .h).
- **Variáveis globais** devem ser evitadas.
- Parte da correção poderá ser feita de forma automatizada, portanto **siga rigorosamente os padrões de saída especificados**, caso contrário sua nota pode ser prejudicada.
- **Legibilidade e boas práticas** de programação serão avaliadas.