Código 10.15 Iniciando Agentes de Pesquisa

```
1  public class Principal extends Agent {
2   ...
3   static AgentController agenteDijkstra, agenteAcs, agenteAs;
4   private static void criandoAgentes() {
5    agenteDijkstra = container.createNewAgent("agenteDijkstra", "Dijkstra", null);
6    agenteAcs = container.createNewAgent("agenteAcs", "AntColonyOptimization", null);
7    agenteAs = container.createNewAgent("agenteAs", "AntColonyOptimization", null);
8    agenteDijkstra.start();
9    agenteAcs.start();
10   agenteAs.start();
11  }
12 }
```

As linhas 5,6 e 7 do Código 10.15 instanciam os agentes Dijkstra, ACS e AS no JADE. Já as linhas 8,9 e 10 deste código realizam o registro dos mesmos no *AMS* do *framework*, identificando-os e deixando-os aptos a realizar suas tarefas de busca.

Para que o agentePrincipal possa receber as mensagens desses outros agentes, no método *setup()* da classe *Principal* é adicionada uma tarefa do tipo *CyclicBehaviour* do JADE, como verificado no Código 10.16.

Código 10.16 Escutando e Encerrando os Agentes de Pesquisa

```
1  public class Principal extends Agent {
2   ...
3   protected void setup() {
4    addBehaviour(new ResponderBehaviour());
5   }
6   public static void atualizaCaminho(String agente) throws Exception {
7    ...
8    new MarcaMenorCaminho(GraphDao.getGrafo(), getMenorCaminho());
9   }
10 }
11 class ResponderBehaviour extends CyclicBehaviour {
12  MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
13  public void action() {
14   ACLMessage aclMessage = myAgent.receive(mt);
15   if (aclMessage != null) {
16    Principal.atualizaCaminho(aclMessage.getSender().getLocalName());
17    myAgent.getContainerController().getAgent("agenteDijkstra").kill();
18    myAgent.getContainerController().getAgent("agenteAcs").kill();
19    myAgent.getContainerController().getAgent("agenteAs").kill();
20   }else {this.block();}
21  }
22 }
```

Uma tarefa do tipo *CyclicBehaviour* executa uma ação ininterruptamente até que seja encerrada. Nesse caso específico, esta ação corresponde a estar sempre ouvindo os agentes de pesquisa, mantendo-se alerta a qualquer mensagem por eles enviada. As mensagens, como definido no projeto da aplicação, serão enviadas pelos agentes assim que os mesmos encontrarem a solução do problema.

A linha 15 do código 10.16 realiza um teste inicial, para verificar se existem mensagens enviadas ao agentePrincipal. Em caso negativo, a tarefa é bloqueada para não consumir parte do processamento da máquina. Não obstante, no momento em que uma mensagem chega, a tarefa é acordada e volta a ser executada. É neste instante que o agentePrincipal, através do método *atualizaCaminho()*, processa a resposta enviada por um dos agentes, marcando graficamente na interface da aplicação o caminho encontrado.

Inicialmente, esse método assinala na interface qual foi o agente responsável pela solução do problema. Em seguida, através da instanciação da classe *MarcaMenorCaminho*, é produzido no componente gráfico da aplicação, a visualização do caminho encontrado pelo agente, através da diferenciação das cores dos vértices e arestas que o compõem. Para a viabilização deste recurso, foi criado o atributo booleano *shortestPath* nas classes que representam estes dois elementos, e através dos métodos *setShortestPath()*, os componentes dos caminhos podem ser assinalados, como mostra o Código 10.17.

Código 10.17 `Marcando o Menor Caminho no Grafo`

```
1  public class MarcaMenorCaminho {
2   public MarcaMenorCaminho(...Graph<MyNode, MyLink> grafo, List<MyNode> caminho) {
3    MyNode no;
4    MyLink edge;
5    MyNode noAnterior = null;
6    Iterator<MyNode> iterVertices = caminho.iterator();
7    while (iterVertices.hasNext()) {
8     no = iterVertices.next();
9     no.setShortestPath(true);
10    if (noAnterior != null) {
11     edge = grafo.findEdge(no, noAnterior);
12     edge.setShortestPath(true);
13    }
14    noAnterior = no;
15   }
16  }
17 }
```

Após a criação gráfica representativa dos caminhos, o agentePrincipal encerra todos os outros agentes ativos na plataforma, como verificado entre as linhas 17 e 19 do Código 10.16. Neste ponto, o Fluzz volta ao seu estado original, mantendo-se preparado para realização de novas buscas.

Como definido em fase de projeto, o agenteDijkstra realiza seu procedimento de busca implementando o Algoritmo de Dijkstra. O JUNG disponibiliza recursos para utilização deste algoritmo através da classe *DijkstraShortestPath*. Seus métodos *getPath()* e *getDistance()* são capazes de retornar respectivamente o menor caminho entre dois vértices informados, e o peso acumulado através do mesmo.

Não obstante, não existe a possibilidade de aplicar algum modelo particular à lógica de avaliação qualitativa dos caminhos encontrados pelo método *getPath()*. As únicas opções de avaliações providas pelo método são de forma exclusiva, ou o grau de separação, ou a distância ponderada entre os vértices. Um modelo um pouco mais complexo, como o modelo *BestWay*, definido no projeto da aplicação, não tem como ser empregado, e por esta razão, a classe de otimização provida pelo *framework* não foi utilizada na implementação do agenteDijkstra no Fluzz.

Com o descarte desse componente do JUNG, outra classe foi desenvolvida para representar o agenteDijkstra, a classe *Dijkstra*, visualizada no Código 10.18.

Código 10.18 `agenteDijkstra`

```
1  public class Dijkstra extends Agent {
2   ...
3   protected void setup() {
4    procuraMenorCaminho();
5   }
6   public void procuraMenorCaminho() {
7    menorCaminho = retornaMenorCaminho(getGrafo(), getNoOrigem(), getListaDestino());
8    Principal.setMenorCaminho(menorCaminho);
9    enviarMensagem();
10   ...
11  }
12  public List<MyNode> retornaMenorCaminho(...Graph... grafo, MyNode noOrigem,
13  List<MyNode> listaDestino, int pesoLacoFraco) {
14   ... //implementa o Algoritmo de Dijkstra e retorna o menor caminho
15   if (degree >= 4) {
16    AntColonyOptimization.setDegree(degree);
17    GeneticAlgorithm.setDegree(degree);
18   }
19  }
20  private void enviarMensagem() {
21   AID r = new AID ("agentePrincipal@" + getHap(), AID.ISGUID);
22   ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
23   aclMessage.addReceiver(r);
24   aclMessage.setContent(this.getLocalName());
25   this.send(aclMessage);
26  }
27 }
```

Ao ser ativado pelo agentePrincipal, o agenteDijkstra, através de seu método *setup()*, invoca o método *procuraMenorCaminho()*. Este último método, por sua vez, é responsável por invocar o método *retornaMenorCaminho()*, que utilizará o Algoritmo de Dijkstra para navegar pela rede, retornando no final o caminho encontrado.

Durante a travessia da rede até a localização do vértice desejado, o agenteDijkstra utiliza-se do modelo avaliativo *BestWay*, proposto no capítulo anterior. Consequentemente, os caminhos vão sendo diferenciados, e o melhor caminho existente entre dois vértices pode ser retornado pelo agente. Para a definição do valor limite que caracterizará um laço fraco na rede, foi criado no painel superior da aplicação Fluzz uma opção onde o usuário pode informar este valor nominalmente.

A linha 16 do código do agenteDijkstra realiza uma importante ação dentro da Sociedade de Agentes de Pesquisa. Somente quando este agente alcança o quarto grau de separação entre os dois vértices pesquisados, é que o mesmo autoriza as formigas artificiais a devolverem suas supostas soluções. Em outras palavras, apesar dos agentes ACS e AS também já estarem ativos e pesquisando pela rede, somente quando o agenteDijkstra ultrapassa a fronteira dos três graus de separação, é que estas formigas podem, caso encontrem uma solução antes do agenteDijkstra, finalizar o processamento.

Caso o agenteDijkstra encontre a solução do problema antes de seus comparsas, ele envia uma mensagem ao agentePrincipal através do método *enviarMensagem()*, apresentado entre as linhas 20 e 26 do Código 10.18. Finalmente, o agentePrincipal recebe a mensagem, e apresenta a solução graficamente na interface do Fluzz.

Os tempos de pesquisa do agenteDijkstra, diante de duas configurações de sua estrutura de dados, e de três redes de tamanhos diferentes, foram cronometrados a fim de se avaliar sua performance. Os dados referem-se ao pior caso encontrado em vinte mensurações para cada rede, e podem ser verificados na Tabela 10.4.

Tabela 10.4: *Performance dos Heaps Perante as Listas*

Estrutura de Dados	10.000 vértices	20.000 vértices	50.000 vértices
ArrayList	1 s	9 s	113 s
Heap	0 s	0 s	4 s

A análise dos dados dessa tabela mostra como a implementação da fila de prioridade do agenteDijkstra através de um *heap*, em vez de uma lista, aumenta significativamente a sua performance. Por isso, a classe *Heap* do pacote *EDU.oswego.cs.dl.util.concurrent.Heap* inerente ao Java 7 foi utilizada para este agente.

Os agentes ACS e AS baseados em formigas artificiais foram implementados através da classe *AntColonyOptimization*, apresentada no Código 10.19. Duas instâncias desta classe representarão ambos os agentes, criados com pequenas diferenças.

O processo de inicialização e encerramento dos agentes ACS e AS é idêntico ao do agenteDijkstra, com a invocação em sequência dos métodos *setup()*, *procuraMenorCa-*

minho(), *retornaMenorCaminho()*, e *enviarMensagem()*. Todavia, o método *retornaMenorCaminho()* ao invés do Algoritmo de Dijkstra, implementa variações dos algoritmos *Ant Colony System* e *Ant System*, para a realização da busca pelos melhores caminhos.

Código 10.19 `agenteACS e agenteAS`

```
1  public class AntColonyOptimization extends Agent {
2   ...
3   public List<MyNode> retornaMenorCaminho(...Graph... grafo, MyNode noOrigem,
4   List<MyNode> listaDestino, int pesoLacoFraco) {
5    while (!isCondicaoParada) {
6     nAnts = 0;
7     startAnt: while (nAnts < 2) {
8      cycles = 0;
9      while (!listaDestino.contains(noPrincipal)) {
10      ... // cria formiga parcial pela busca em profundidade com backtracking
11      if (getLocalName().equals("agenteAcs")) {
12       evaporaFeromonio(noPrincipal, nodeEscolhido);
13      }
14      cycles ++;
15      if (cycles > 100) {continue startAnt;}
16     }
17     GeneticAlgorithm.setAntsColony(getAntsColony());
18     GeneticAlgorithm.setAntsColonyBest(getAntsColonyBest());
19    }
20    if (getLocalName().equals("agenteAcs")) {
21     somaFeromonioArestasPercorridas(null, getBestAnt());
22    }else if (getLocalName().equals("agenteAs")) {
23     somaFeromonioArestasPercorridas(ants, null);
24    }
25    if (!isAgenteGa()) {createAgentGa();}
26   }
27  }
28 }
```

A linha 5 do Código 10.19 é responsável por manter a busca em execução até que a solução seja encontrada. Para isso, diversas formigas artificiais, que representam os caminhos na rede, serão criadas sequencialmente. O laço especificado na linha 7 deste código existe para que após a criação da segunda formiga, o feromônio possa ser atualizado nas arestas, como mostra o código entre as linhas 20 e 24.

Os agentes ACS e AS representam colônias de formigas diferentes. Todavia, foi criada somente uma única trilha de feromônio a ser compartilhada por ambas as colônias, que, portanto, compartilharão de seus aprendizados. O atributo *pheromone* da classe *MyLink* é o responsável por manter o feromônio depositado pelas formigas.

Outra característica importante, é que ambos os agentes divergem sobre a forma de manipulação do feromônio. O agenteACS realiza o seu depósito de forma elitista, o que significa que o procedimento é feito somente pela melhor formiga gerada pelo agente. Já o agenteAS permite que todas as suas formigas participem do processo, sendo o depósito de feromônio realizado de maneira coletiva. Quanto à evaporação, somente o agenteACS realiza este procedimento, processando uma pequena redução da substância sobre as arestas percorridas por suas formigas.

Para o processo de criação de formigas, iniciado no laço da linha 9 do Código 10.19, foi estipulado um valor limite de 100 ciclos, que na prática significam 100 vértices. Em outras palavras, nenhum caminho pode ser criado com mais de 100 vértices, e se isto ocorrer, a formiga deverá ser reinicializada como feito na linha 15 deste código.

Esse processo de geração de caminhos consiste basicamente no *DFS - Depth First Search* [32], ou algoritmo de busca em profundidade em grafos, sendo que as formigas utilizam-se de uma lista tabu para verificarem os vértices já visitados. Caso a formiga chegue a um beco sem saída, ela pode realizar a operação conhecida como *backtracking* [32] ou retrocesso, voltando para a última configuração válida do caminho.

Somente a informação do feromônio foi utilizada como fator de orientação das formigas, já que a informação heurística, o peso das conexões, foi desconsiderada. Como as formigas possuem suas zonas de trabalho além dos três graus de separação, este peso não se torna a priori algo relevante, e o processo pode ser agilizado. Além disso, o agenteACS não faz uso do mecanismo de explotação explícita proposto originalmente para o seu algoritmo, se concentrando na diversificação de suas soluções.

Por fim, as linhas 17 e 18 do Código 10.19 são responsáveis por enviar ao último agente a ser ativado na Sociedade, o agenteGA, todas as formigas geradas anteriormente. Com o término da produção das duas primeiras formigas, o agenteGA é então inicializado na linha 25 deste código, passando a integrar os agentes de pesquisa.

A classe *GeneticAlgorithm* apresentada no Código 10.20 é responsável por criar o agente que implementará um Algoritmo Genético para a realização de buscas. A ativação deste agente segue o mesmo processo de invocação de métodos utilizado por todos os agentes anteriores, e o processo de utilização de suas soluções, segue a mesma regra dos agentes ACS e AS. Somente quando o agenteDijkstra ultrapassa a fronteira dos três graus de separação, o agenteGA está apto a encerrar o processamento, caso encontre uma solução antes de seus comparsas. A linha 17 do Código 10.18, representa o momento em que o agenteDijkstra informa o agenteGA sobre esta ocorrência.

Como definir indivíduos para a população inicial do Algoritmo Genético não é uma tarefa simples, se o problema em questão for encontrar caminhos válidos nas redes, o mais prudente seria utilizar as próprias formigas artificias outrora criadas. Sendo assim, o método *searchAnts()* invocado na linha 5 do Código 10.20 é responsável por pesquisar

formigas armazenadas em um depósito populado pelos agentes ACS e AS.

Esse depósito é dividido em três seções: a seção geral, que compreende todas as formigas recebidas, a seção elitista, que armazena as melhores formigas, e a seção mutante, que contempla as formigas geneticamente modificadas. Entre as linhas 11 e 18 do Código 10.20, é apresentado o momento em que o agenteGA realiza a seleção das formigas. Como as três seções do depósito possuem a mesma chance de serem escolhidas, e como as duas últimas mencionadas possuem relativamente poucas formigas, há uma grande chance de todas estas formigas serem recombinadas, o que garante, de certa forma, um processo de recombinação elitista.

Código 10.20 `agenteGA`

```
1  public class GeneticAlgorithm extends Agent {
2   ...
3   public List<MyNode> retornaMenorCaminho() {
4    while (!isCondicaoParada) {
5     antList = searchAnts(2);
6     ... // realiza operações genéticas nas formigas
7    }
8   }
9   private List<List<MyNode>> searchAnts(int nAnts) {
10   ...
11   if (random <= 33) {
12    randomAnt = getAntsColonyBest().get(new Random().nextInt(maxSize));
13   }else if (random > 33 && random <= 66) {
14    randomAnt = getAntsColony().get(new Random().nextInt(getAntsColony().size()));
15   }else if (random > 66 && random <= 100) {
16    randomAnt = getAntsColonyCross().get(
17     new Random().nextInt(getAntsColonyCross().size()));
18   }
19  }
20 }
```

O laço criado na linha 4 do Código 10.20 mantém o agente em funcionamento até que uma solução seja encontrada. Assim que duas formigas são selecionadas no depósito, começam a ser recombinadas através do processo *NBX (Node Based Crossover)*, que viabiliza o cruzamento somente se os dois indivíduos tiverem pelo menos um par do mesmo gene, desconsiderando os vértices de origem e de destino. Caso os indivíduos escolhidos não sejam compatíveis, outras formigas serão selecionadas para o cruzamento.

Finalizado o cruzamento, as formigas ainda podem ser submetidas a outra operação genética: a mutação. Com probabilidade de 1% de ocorrência em cada gene, caso esta operação seja realizada, a troca do material genético é efetivada com a consulta da lista de vértices adjacentes do gene anterior ao escolhido. Desta forma, um caminho

parcial válido é criado, sendo posteriormente completado pelo material genético de outra formiga compatível, a partir do novo gene que substituiu o antigo.

A Figura 10.12 ilustra o processo de busca da aplicação Fluzz. Redes geradas pelos quatro modelos de rede, Erdös/Rényi, Watts/Strogatz/ Barabási/Albert e Marin/Carvalho, e dispostas com 100 vértices criados com duas conexões por iteração, foram utilizadas para a exemplificação do procedimento.

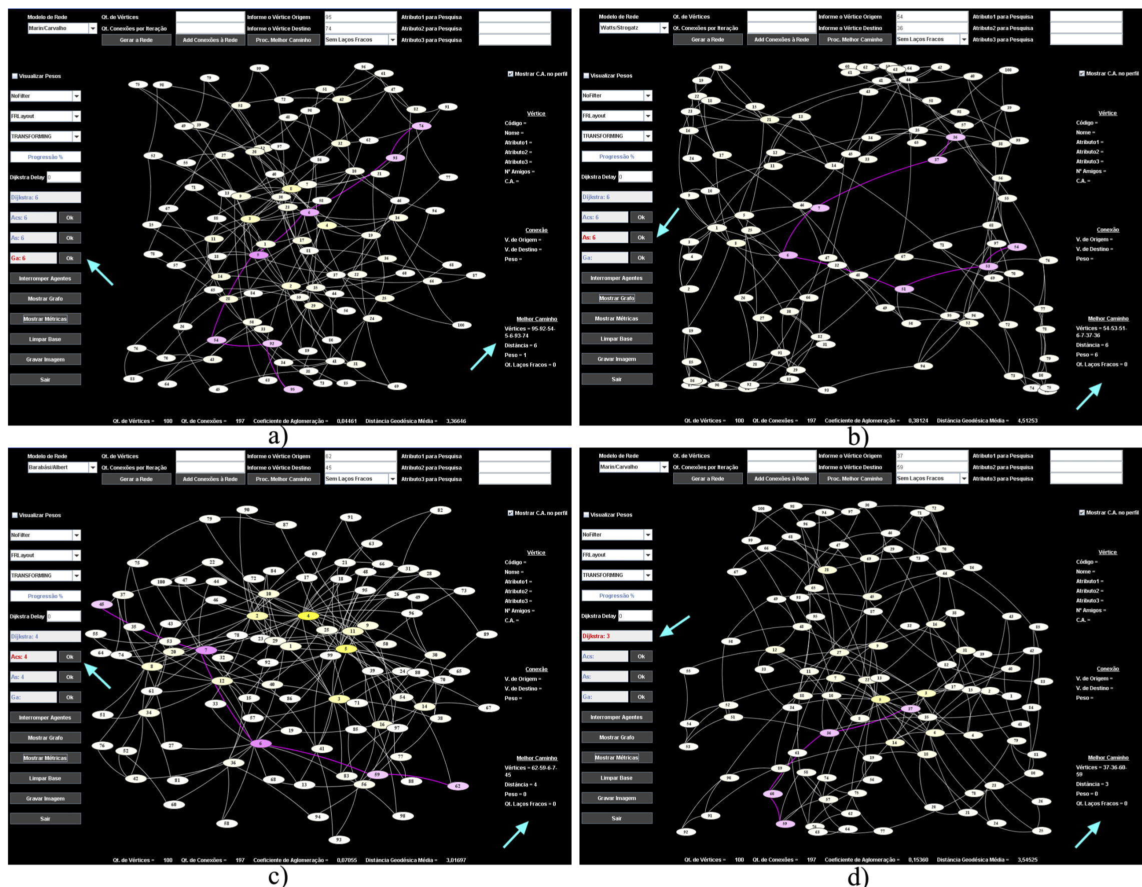

Figura 10.12: *Processo de Busca.*

Na posição (a) dessa figura, o agenteGA foi o responsável pela solução do problema. Como este agente conseguiu encontrar um caminho entre os vértices pesquisados antes dos outros agentes, ele interrompeu o processamento e devolveu a solução. Já na posição (b) da figura, o agenteAS obteve a melhor performance, apresentando a solução. Na posição (c) da figura, pode ser observado que desta vez o agenteACS foi o vitorioso.

Todas essas soluções apresentadas por esses três agentes metaheurísticos só foram possíveis, pois o agenteDijkstra já concentrava suas buscas em um nível de distância superior aos três graus de separação. Como os agentes ACS, AS e GA já haviam encontrado uma solução no último nível pesquisado pelo agenteDijkstra antes de sua interrupção, eles puderam apresentá-la primeiramente, o que não ocorreu com a exemplificação na posição (d) da Figura 10.12, onde o agenteDijkstra foi o vencedor.

O Fluzz possui um recurso visual em seu painel esquerdo para a identificação do agente responsável pelas soluções encontradas. Conforme assinalado na Figura 10.12, a aplicação destaca o espaço destinado à solução do agente vitorioso, através de uma coloração em vermelho capaz de distingui-lo dos demais.

Além disso, é nesse espaço que as soluções parciais dos agentes vão sendo apresentadas, sendo possível verificar o grau de separação em que cada um se encontra. Enquanto o agenteDijkstra tem sua distância na rede aumentada, à medida que o mesmo ultrapassa níveis de distância, os agentes ACS, AS e GA têm suas distâncias diminuídas, à medida que seus caminhos vão sendo melhorados.

No painel direito da aplicação, existe um espaço destinado à descrição detalhada dos caminhos encontrados, apresentando informações como a identificação dos vértices que os compõem, o grau de separação entre os vértices de origem e destino, o peso acumulado, e a quantidade de laços fracos presentes.

Além da descrição nominal dos vértices, o Fluzz ainda detém um importante recurso para a visualização das soluções retornadas pelos agentes. Assim que são encontrados, os caminhos são realçados graficamente no componente de renderização visual da aplicação, que é capaz de assinalar os vértices e arestas compreendidos, através de uma tonalidade roxa. Este processo, como pode ser visto em todas as imagens da Figura 10.12, é fundamental para a localização visual dos componentes que integram os caminhos.

Algumas mensurações foram realizadas para comparar as performances dos agentes durante o procedimento de busca. A Tabela 10.5 apresenta os valores encontrados durante vinte medições para duas variações do modelo de rede Marin/Carvalho, sendo a primeira criada com duas conexões a cada iteração, e a segunda criada com cinco.

Tabela 10.5: *Performance dos Agentes de Pesquisa*

	N º Vezes que os Agentes ACS, AS e GA Foram Vitoriosos em 20 Medições	
Qt. Vértices	Rede Marin/Carvalho (2)	Rede Marin/Carvalho (5)
1000	7x	5x
2000	6x	4x
5000	3x	1x
100000	2x	1x
200000	1x	1x
500000	0x	0x

Os dados mensurados significam a quantidade de vezes, dentre as vinte medições, que os agentes ACS, AS ou GA foram responsáveis pela determinação da solução. Como pode ser observado na tabela, à medida que a rede torna-se maior ou mais densa, estes agentes têm suas performances prejudicadas, sendo o agenteDijkstra cada vez mais vitorioso. Isso ocorre basicamente por existirem mais caminhos na rede, o que aumenta a chance dos agentes metaheurísticos se perderem por um tempo, até começarem a convergir em definitivo para uma boa solução.

Além disso, a estrutura de dados criada para o agenteDijkstra foi fundamental para seus resultados de performance, consequentemente proporcionando a este agente a grande maioria das soluções. Não obstante, os agentes metaheurísticos, além de serem responsáveis por algumas das soluções, são capazes de prover um importante mecanismo para a aplicação Fluzz: a diversificação.

Como o agenteDijkstra retorna sempre o mesmo caminho para uma busca repetida entre dois vértices, não existe a possibilidade de diversificação das soluções. É neste ponto que a natureza estocástica dos agentes metaheurísticos torna-se extremamente valiosa. Em uma mesma pesquisa realizada diversas vezes, os agentes ACS, AS e GA são capazes de retornar diferentes soluções, caso as mesmas existam na rede.

Dessa forma, foi criado um recurso para que os agentes metaheurísticos tenham mais chances de interromper o agenteDijkstra. No painel esquerdo do Fluzz, existe um campo para se informar um tempo de *delay*, ou espera em milisegundos, que o agenteDijkstra deverá utilizar ao avançar um nível de distância na rede. Este procedimento torna o agenteDijkstra mais lento, dando mais oportunidades para os outros agentes.

Apesar de parecer estranho, querer prejudicar propositalmente a performance do agenteDijkstra, este fato viabiliza a utilização de um outro artifício da aplicação. Botões foram criados ao lado de cada agente metaheurístico, e quando acionados, interrompem o processamento da pesquisa, que devolverá a solução atual do respectivo agente. Caso o agenteACS, por exemplo, tenha encontrado um caminho de qualquer distância, e o usuário quiser conferi-lo, basta que seja efetivado um clique do botão correspondente, para que a aplicação encerre o processamento e devolva o caminho referente.

Esse recurso funciona independentemente da posição em que o agenteDijkstra se encontra, e serve para que diferentes soluções de uma mesma pesquisa, possam ser repassadas pelos agentes metaheurísticos. Em termos práticos, correntes de amizade alternativas podem ser importantes para a pessoa de origem, quando um dos indivíduos pertencentes a determinado caminho retornado, cria alguma barreira comunicacional.

Todas as redes apresentadas na Tabela 10.2 foram utilizadas para a realização de buscas entre diferentes vértices. De forma geral, a Sociedade de Agentes de Pesquisa processou as informações corretamente, produzindo os resultados esperados de acordo com os modelos propostos, e mantendo um tempo médio de busca de poucos segundos.

10.2.25 RF: Detecção de caminhos entre uma pessoa e várias outras

O Fluzz também provê recursos para que buscas sejam realizadas de forma $1:N$, o que significa ter vários indivíduos sendo alvos simultaneamente em uma única pesquisa.

Existem no painel superior do Fluzz, três campos preparados para receberem valores referentes aos atributos configuráveis do modelo relacional, criado na fase de

projeto. Estes atributos podem representar qualquer propriedade inerente aos indivíduos formadores das redes, como características pessoais, profissionais, dentre outras.

Para que esse recurso seja utilizado, é necessário informar nos campos mencionados, pelos menos um desses atributos configuráveis, ao invés do código do vértice de destino. Neste momento, o agentePrincipal identifica que a busca deve contemplar todos os vértices que possuem os atributos informados.

O método responsável pela identificação desses indivíduos é o *retornaNosPesquisa()* da classe *GraphDao*, como mostrado na linha 10 do Código 10.21. Este código pertence ao evento invocado com o clique do botão *Proc. Melhor Caminho*, que é a ação esperada pelo agentePrincipal para iniciar o procedimento de busca.

Código 10.21 Definindo Alvos Para Pesquisa

```
1  public class Principal extends Agent {
2   ...
3   btnMenorCaminho.addActionListener(new ActionListener() {
4    public void actionPerformed(ActionEvent event) {
5     ...
6     if (!txtNoDestino.getText().equals("") && isDigit(txtNoDestino.getText())) {
7      listaNosDestino.add(GraphDao.getMapaNosBanco().get(
8       Integer.parseInt(txtNoDestino.getText())));
9     }else {
10     listaNosDestino = new GraphDao().retornaNosPesquisa(noOrigem,txtNoDestino.getText(),
11      txtAtributo1.getText(),txtAtributo2.getText(),txtAtributo3.getText());
12    }
13   }
14  });
15 }
```

Como observado na linha 6 desse código, o sistema verifica se o código do vértice de destino foi informado, e em caso afirmativo, realiza a busca de forma 1 : 1, procurando somente por um alvo na rede. Caso o código do vértice de destino não tenha sido informado, o método *retornaNosPesquisa()* é invocado, retornando uma lista de indivíduos que possuem os atributos inseridos pelo usuário nos campos de pesquisa.

Consequentemente, essa lista de indivíduos é enviada para a Sociedade de Agentes de Pesquisa, que realizará o procedimento de busca mantendo-os todos como alvos. O melhor caminho encontrado pela Sociedade, dentre todos os indivíduos dessa lista, é então retornado pela aplicação.

Uma particularidade ocorre quando o usuário informa o nome ao invés do código, no campo referente ao vértice de destino. Caso o sistema encontre mais de um indivíduo com o nome informado, a pesquisa torna-se de 1:N, e o mesmo procedimento detalhado anteriormente, é então realizado.

10.2.26 RF: Criação de filtros para visualização dos caminhos

Como as redes geradas pelo Fluzz podem possuir grande quantidade de vértices e conexões, a renderização completa do grafo referente pode ser extremamente lenta. Obviamente, a capacidade do *hardware* onde a aplicação estará rodando influencia na quantidade de elementos passíveis de ser visualizados e manipulados pelo usuário, sem que haja um comprometimento da performance do mesmo.

É por este motivo que a aplicação não apresenta automaticamente a visualização dos grafos após o procedimento de geração de redes, sendo necessário o usuário explicitar esta necessidade através do clique do botão *Mostrar Grafo*. Desta forma, uma rede bastante extensa como a apresentada na Tabela 10.2 de 500.000 vértices, que se visualizada comprometeria o desempenho do *hardware*, pode ser obtida rapidamente no banco de dados e disponibilizada normalmente para a realização do procedimento de busca.

Consequentemente, as soluções providas pela Sociedade de Agentes de Pesquisa vão sendo devolvidas e renderizadas através de filtros, que serão responsáveis por selecionar somente os vértices relevantes para a pesquisa, descartando todos os demais. Com isso, subgrafos da rede original são apresentados pelo agentePrincipal representando a solução do problema, conforme apresentado na Figura 10.13.

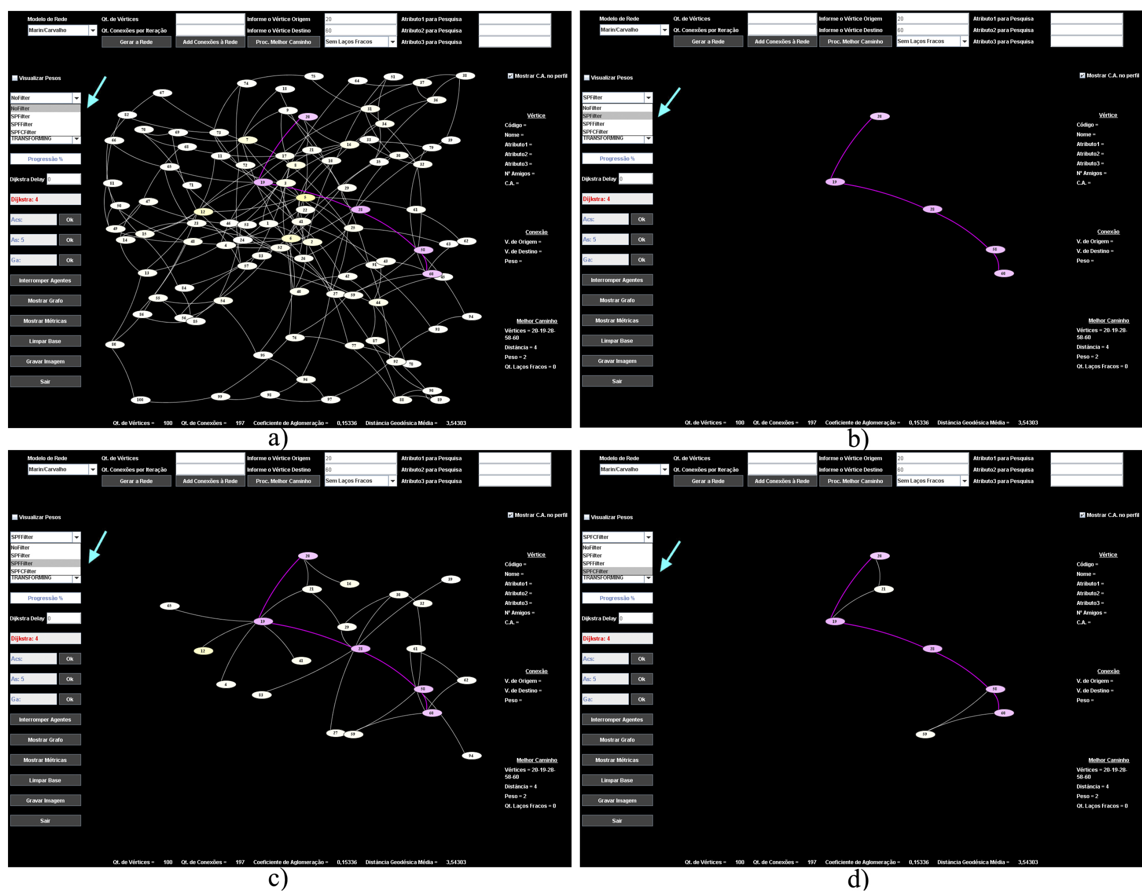

Figura 10.13: *Processo de Filtragem de Soluções.*

Três diferentes tipos de filtros foram implementados no Fluzz. Cada um realiza um procedimento específico de filtragem de vértices e arestas dos grafos, sempre tendo o grafo original e a solução devolvida pelos agentes como valores de entrada.

Na posição (a) Figura 10.13, a visualização completa de um grafo com 100 vértices foi provida pela aplicação. Pode-se verificar que o procedimento de busca foi realizado nesta rede, com o agentePrincipal destacando o caminho de quatro graus de separação encontrado pelo agenteDijkstra. No painel esquerdo do Fluzz existe uma opção para que filtros possam ser aplicados nos caminhos encontrados pelos agentes de pesquisa. Como nessa primeira imagem, a opção *NoFilter* foi selecionada, nenhum filtro foi aplicado ao grafo, que teve todos os seus elementos apresentados.

Na posição (b) dessa figura, a opção de filtro *SPFilter (Shortest Path Filter)*, ou filtro do menor caminho, foi selecionada. Sendo assim, somente os vértices e arestas pertencentes ao caminho devolvido pelo agente vitorioso na pesquisa são apresentados. Outra opção de filtro é a *SPFFilter (Shortest Path Friends Filter)*, ou filtro do menor caminho com amigos, utilizado na posição (c) da mesma figura. Percebe-se que além dos componentes pertencentes ao caminho encontrado, foram apresentados todos os amigos de cada vértice que compõe este caminho.

Por fim, foi criada a opção de filtro *SPFCFilter (Shortest Path Friends in Common Filter)*, ou filtro do menor caminho com amigos em comum. Como apresentado na posição (d) da Figura 10.13, este filtro além de apresentar os vértices e arestas do caminho encontrado, adiciona ao subgrafo os amigos existentes em comum entre quaisquer dois vértices pertencentes ao caminho.

Para redes com mais de 100 vértices, o resultado do procedimento de busca sempre é apresentado com a opção de filtro *SPFilter* acionada, por ser a que menos desenha componentes na tela, agilizando o retorno da solução. Caso o usuário deseje visualizar a solução por um dos outros filtros, ou até mesmo pelo grafo inteiro, deve proceder a alteração desta opção na interface da aplicação.

Dois recursos técnicos são utilizados pelo Fluzz para a realização de filtros. O primeiro utiliza-se da criação de predicados aos vértices, através da classe *Graph_VertexPredicate*, apresentada no Código 10.22. Estes predicados são utilizados somente para os grafos que possuem 100 vértices ou menos, e representam a adequação destes componentes aos critérios de filtragem inerentes a cada filtro.

Como pode ser visto no código, essa classe implementa a interface *Predicate*, que impõe à mesma a necessidade de sobrescrita do método *evaluate()*. É dentro deste método que as condições de apresentação de cada vértice são analisadas, seguindo a lógica estabelecida por cada filtro. Caso o vértice deva ser apresentado na interface, o método retorna um valor booleano *true*, e em caso contrário, *false*. Um detalhe importante é que as arestas existentes entre os vértices selecionados são sempre apresentadas.

Código 10.22 Criando Predicados aos Vértices

```
1  public class Graph_VertexPredicate<V,E> implements Predicate<Context<Graph<V,E>,V>> {
2   ...
3   public boolean evaluate(Context<Graph<V, E>, V> context) {
4    ... // de acordo com a lógica de cada filtro, retorna true para um vértice
5     a ser apresentado e false caso contrário
6   }
7  }
```

O método responsável por aplicar um predicado aos vértices é o *setVertexIncludePredicate()* da *API* do JUNG. Como pode ser visto nas linhas 7 e 8 do Código 10.23, ao ser invocado o evento referente à escolha de um dos filtros na interface, uma instância da classe *Graph_VertexPredicate()* é criada, informando qual filtro será aplicado.

Código 10.23 Aplicando Filtros às Redes

```
1  public class Principal extends Agent {
2   ...
3   cboFiltro.addActionListener(new ActionListener() {
4    public void actionPerformed(ActionEvent event) {
5     if (GraphDao.getGrafo().getVertexCount() <= 100) {
6      if (cboFiltro.getSelectedItem().equals("SPFilter")) {
7       component.getRenderContext().setVertexIncludePredicate(
8        new Graph_VertexPredicate<MyNode, MyLink>(true, "SPFilter"));
9      }
10     ... // testa e aplica outros tipos de filtro
11    }else {
12     if (cboFiltro.getSelectedItem().equals("SPFilter")) {
13      subGrafo = new Graph_FilterGraph().getSubGraph(GraphDao.getGrafo(),
14       getMenorCaminho(), "SPFilter");
15      mostraGrafo(subGrafo);
16     }
17     ... // testa e contrói outros grafos de acordo com os filtros
18    }
19   }
20  });
21 }
```

O segundo procedimento para a realização de filtros consiste na criação de um novo grafo que representará um subgrafo do grafo original. Desta forma, não existe aplicação de predicados aos vértices, pois um novo grafo propriamente dito deve ser gerado. Não obstante, este novo grafo deve apresentar a mesma topologia que obteria, caso o filtro tivesse sido realizado por meio de predicados.

Esse recurso é aplicado a grafos com mais de 100 vértices, e foi criado especialmente pela possibilidade de existência de redes extensas e densas. Estas redes certamente prejudicariam a performance da aplicação, inclusive na composição visual dos subgrafos provenientes dos filtros, já que o JUNG sempre mantém todo o grafo na memória. Assim, grafos menores são criados de acordo com as regras estabelecidas pelos filtros.

A linha 13 do Código 10.23 mostra o momento em que um novo grafo é gerado para simular o filtro. O método *getSubGraph()* da classe *Graph_FilterGraph*, visualizada no Código 10.24, é invocado tendo como objetivo produzir um subgrafo, a partir do grafo original e das características do filtro escolhido. Em seguida, o método *mostraGrafo()* da classe *Principal* é acionado para representar visualmente o subgrafo criado.

Código 10.24 `Criando Subgrafos`

```
1 public class Graph_FilterGraph {
2   ...
3   public UndirectedSparseGraph<MyNode, MyLink> getSubGraph(...Graph... originalGraph,
4     List<MyNode> listaNos, String tipoFiltro) {
5     ... // de acordo com a lógica de cada filtro, cria um novo grafo para os vértices
6       que devem ser apresentados, devolvendo-o como retorno
7   }
8 }
```

10.2.27 RF: Salvamento da imagem da rede gerada

As imagens das redes geradas pela aplicação podem ser salvas em disco, através de arquivos no formato *jpg*. No painel esquerdo do Fluzz, existe um botão *Gravar Imagem*, que ao ser acionado invoca a classe *GravarImagemGrafo*, responsável pela criação do arquivo *graph.jpg*. Este arquivo é produzido de acordo com a imagem definida no painel central da aplicação, e é salvo automaticamente no diretório corrente da mesma.

10.2.28 RF: Deleção da base de dados

Caso exista a necessidade de exclusão das informações da base de dados, o botão *Limpar Base*, localizado no painel esquerdo da aplicação, deve ser acionado. Neste instante, a classe *ApagarGrafo* é instanciada pelo agentePrincipal, procedendo a deleção dos dados e a destruição do grafo anteriormente criado.

10.2.29 RNF: Modularidade

A modularidade da aplicação foi obtida através da implementação de agentes de *software*, que interagem entre si visando a resolução cooperativa dos problemas. O

framework JADE foi utilizado para a criação deste sistema multiagente, capaz de operacionalizar diversas atividades por meio de agentes diferentes, com funções específicas.

10.2.30 RNF: Simplicidade

A criação da Sociedade de Agentes de Pesquisa possibilitou à aplicação, a possibilidade de receber futuramente novos agentes capazes de contribuir com suas características particulares, bastando para isso, que eles sejam incorporados à mesma.

10.2.31 RNF: Paralelismo

A utilização de múltiplos agentes pelo Fluzz garante o processamento paralelo das rotinas executadas, melhorando significativamente a performance da aplicação.

10.2.32 RNF: Eficiência

A aplicação construída é capaz de interpretar e processar cada modelo proposto, seja no tocante à geração de redes, ou no procedimento de busca entre os vértices, retornando graficamente, soluções condizentes com as características particulares definidas.

10.2.33 RNF: Confiabilidade

A arquitetura da aplicação propicia que os procedimentos de busca sejam sempre finalizados, independentemente da falha de algum dos agentes de pesquisa, sendo o agentePrincipal responsável por encerrar agentes interrompidos indevidamente.

10.2.34 RNF: Portabilidade

Como foi todo implementado em Java, o Fluzz torna-se exequível independentemente de qual seja o sistema operacional ou o processador que executa seu código.

10.2.35 RNF: Manutenibilidade

A fim de aumentar a legibilidade e facilitar a manutenção do sistema, o padrão de projeto MVC (Modelo-Visão-Controle), responsável por dividir o *software* em camadas funcionais, foi utilizado na implementação da aplicação.

10.2.36 RNF: Usabilidade

A interface do Fluzz foi concebida de forma a facilitar a compreensão de seus recursos, fazendo com que os mesmos sejam utilizados pelos usuários de forma instintiva.

CAPÍTULO 11

Considerações Finais

Esta dissertação apresentou um estudo embasado em diversas áreas do conhecimento, que convergiram para que o projeto da aplicação Fluzz fosse efetivado. A Figura 11.1 ilustra como estes campos de pesquisa se relacionam com a ferramenta produzida.

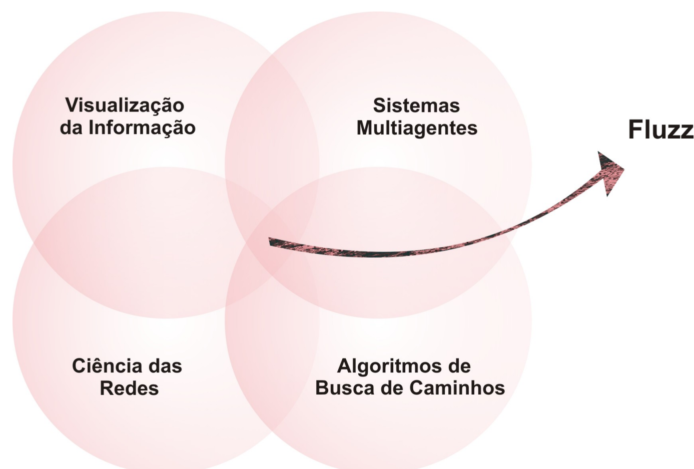

Figura 11.1: *Áreas Envolvidas no Projeto da Aplicação Fluzz.*

Indubitavelmente, o enfoque do conteúdo explorado neste trabalho foi destinado à análise evolutiva das redes encontradas por trás de sistemas complexos, em especial, a sociedade. Ao visualizar apenas nós e *links*, abstraindo todos os outros detalhes supérfluos, adquiriu-se o privilégio de observar a arquitetura da complexidade, vislumbrando os princípios organizacionais universais que formam esses sistemas.

As modelagens e técnicas computacionais apresentadas durante a criação da aplicação foram empregadas com a intenção de utilizar as propriedades das redes a favor dos seres humanos. Obviamente, a compreensão dos padrões evolutivos e organizacionais de cada tipo de rede foi o primeiro passo dado, seguido posteriormente pela disposição de métodos capazes de influenciá-los.

O fino enredamento dos seres humanos atualmente em um único e imenso tecido aberto, interativo e cooperativo, gera uma situação absolutamente inédita e portadora de expectativas para toda a sociedade. As redes sociais *online*, por exemplo, estão se tornando verdadeiras plataformas que seguramente ajudarão os indivíduos a alcançarem

seus objetivos, e prover recursos capazes de viabilizar a cooperação entre os mesmos, ajudará a humanidade a se tornar maior do que a soma de suas partes.

É principalmente dentro desse contexto social que a aplicação Fluzz emergiu, apresentando recursos projetados para que pessoas possam usufruir de autodescobertas, e da possibilidade de realmente se conhecerem e se influenciarem em rede.

A seguir serão abordadas as contribuições realizadas por esta pesquisa, sendo posteriormente apresentadas as limitações da ferramenta desenvolvida, e possíveis desafios a serem enfrentados em trabalhos futuros.

11.0.37 Contribuições

As metas da aplicação construída neste projeto basicamente definem as principais contribuições engendradas pelo mesmo. Como se mostrou eficiente na implementação de todos os seus requisitos funcionais, o Fluzz proveu funcionalidades que podem aproximar seres humanos, lhes dando a capacidade de cooperar em uma escala muito maior do que a experimentada anteriormente.

As técnicas de visualização de rede empregadas no projeto permitem que as pessoas possam encontrar sentido e simplicidade em meio à desordem da complexidade. Em um contexto que concebivelmente poderia ser aleatório e desprovido de qualquer ordem discernível, os indivíduos podem descobrir através da ferramenta, informações ocultas nos tentáculos das redes, e as usarem a seu favor.

Em um mundo abrangido em apenas seis graus, o que acontece em determinado local chega até as pessoas muito mais rápido do que elas pensam. Assim, simplesmente porque algo parece distante em um dado momento, não significa que seja irrelevante ou inacessível. Pequenas mudanças na topologia da rede podem transformar abruptamente, algo visualizado antes como muito distante, em algo incrivelmente próximo.

Prover recursos, capazes de realizar o mapeamento visual das complexas redes sociais em que os indivíduos se mantêm incrustados, torna a aplicação Fluzz útil para uma observação contínua dos padrões evolutivos da sociedade.

Em alguns casos, pode ser interessante saber se uma rede contém um caminho curto que conecta determinado par de indivíduos, ou que alguns indivíduos são muito mais conectados do que os demais. Mas, em outros casos, talvez o importante seja verificar se as pessoas estão localizadas em aglomerados localmente reforçados. É neste sentido, que a aplicação Fluzz disponibiliza seus recursos de mapeamento gráfico, possibilitando que as pessoas reconheçam padrões visuais de organização e comportamentos.

A apresentação de um novo modelo de rede também foi uma importante contribuição deste projeto. O valor deste resultado não está apenas na identificação de um novo tipo de arquitetura, que é apreciado por suas diferenças perante as concepções anteriores

a respeito de redes, e sim em como este ferramental pode favorecer em termos práticos a compreensão da sociedade interconectada.

A proposição das redes de pequena escala, referenciadas na aplicação como Marin/Carvalho, levou a uma espécie de teoria unificada das redes anteriormente propostas na literatura. Ela mescla o pensamento de Barabási e Albert com a teoria de Watts e Strogatz, e encaixa ambas as ideias como peças de um quebra-cabeça, formando uma imagem maior e mais coerente da topologia da sociedade. Consequentemente, este novo modelo de rede pode ser utilizado para a pesquisa sobre a dinâmica evolutiva das redes sociais, contribuindo para uma melhor compreensão deste processo.

As técnicas de busca empregadas pela aplicação embasam-se em estudos sociológicos, que visam criar mecanismos sinérgicos capazes de colocar determinadas pessoas lado a lado. A intenção foi de proporcionar a coleta de informações de fontes específicas, através da formação de conexões estratégicas que potencializariam a emergência de sólidos relacionamentos cooperativos.

Ao se unirem, as pessoas, assim como as formigas, criam algo que transcende o indivíduo, em uma escala superlinear. Como metaforicamente ilustrou Steven Johnson: "uma formiga isolada, que encontrou determinado caminho para um pote de açúcar distante do formigueiro, remonta à imagem do astronauta na Lua: ambos os feitos são permitidos pelos esforços coordenados de muitos indivíduos" [60]. Neste contexto, o Fluzz provê mecanismos inteligentes de busca de prováveis parceiros em rede, que devem trabalhar juntos para alcançar o que não podem realizar sozinhos.

Modelos capazes de avaliar os melhores caminhos entre as pessoas, para os quais são maiores as chances de influência recíproca, tornam a atividade de conexão muito mais provável no ambiente do Fluzz. Agentes de *software*, baseados em algoritmos exatos e metaheurísticos, exploram simultaneamente as redes analisadas, cruzando e analisando vértices e arestas na tentativa de proceder a melhor solução.

11.0.38 Limitações e Trabalhos Futuros

As limitações observadas durante a confecção do Fluzz compreendem basicamente a questão da geração e do processamento visual de redes. Para grafos extensos e densos, foi verificada uma limitação computacional referente, tanto ao *hardware* empregado, quanto ao *framework* gráfico utilizado para a criação dos mesmos.

A ideia inicial e diferencial de confecção da aplicação foi criar redes da ordem de 100 à 1.000.000.000 de vértices, para que fosse possível a realização de buscas nestes ambientes, simulando situações próximas a uma pesquisa realizada na maior rede social *online* atualmente: o Facebook. Não obstante, a capacidade do *hardware* utilizado no

projeto não atendeu à demanda, e o valor limitante para o tamanho das redes se concentrou em 500.000 vértices, representando 0,05% do esperado.

Além disso, a visualização completa das redes processadas implicou em um grande consumo computacional. Com o *hardware* utilizado, apresentado na seção 10.1.8, a performance do computador tornou-se visivelmente abalada durante o processamento gráfico das redes por intermédio do JUNG, mesmo quando a ordem de seus tamanhos ultrapassava ligeiramente a barreira dos 1000 vértices. A utilização de diversos filtros pelo Fluzz também foi uma forma de fornecer uma alternativa ao problema.

Isso mostra que ainda existe um longo caminho a ser trilhado no contexto de geração e visualização de redes reais. Não obstante, a escalabilidade vertical, que compreende a adição de recursos computacionais ao *hardware* como memória ou processador, não deve ser a única preocupação dos interessados no assunto. Certamente, produzir recursos que se utilizem cada vez menos do *hardware* empregado, pode ser uma solução mais barata e mais eficiente de se alcançar o mesmo objetivo.

Como redes com mais de 500.000 vértices foram inviáveis de ser criadas, os tempos de busca cronometrados estiveram dentro das expectativas. Certamente, com mais recursos computacionais empregados, redes maiores poderão ser geradas e novas medições poderão apresentar performances não tão convidativas dos agentes de pesquisa. Consequentemente, melhorias nos modelos ou nas técnicas utilizadas por estes agentes deverão ser providas para oferecerem maior eficiência.

A utilização de bancos de dados relacionais como o PostgreSQL pode também ser vista como uma barreira à escalabilidade da aplicação. Os chamados bancos de dados NoSQL representam uma real alternativa para a adequação da base de dados a uma estrutura escalável horizontalmente, capaz de crescer de forma a acompanhar o crescimento, muitas vezes gigantesco, das redes analisadas.

Finalmente, todos os modelos, técnicas, algoritmos ou arquitetura, pertencentes à aplicação poderão ser empregados livremente em redes sociais *online*, que se beneficiarão dos recursos providos por estes elementos. O objetivo é que estes *sites* de redes sociais, principalmente os mais populares, que detêm grande quantidade de pessoas interconectadas, possam prover recursos de visualização gráfica, bem como de buscas inteligentes, que certamente contribuiriam para os processos cognitivo e interativo humanos.

Como bem afirmou Duncan Watts em seu livro *Six Degress*, ao relatar sobre a importância atual da pesquisa sobre o universo interconectado: "... mais de sessenta anos depois de homens como Rapoport e Erdös começarem com os primeiros estudos, pode ser que a batalha esteja começando a virar a favor da ciência" [106].

Referências Bibliográficas

[1] **British Broadcasting Corporation (BBC)**. Disponível em http://www.bbc.co.uk/news/technology-19816709, acessado em outubro de 2012.

[2] **Dreamstime (Neurônios)**. Disponível em http://pt.dreamstime.com/imagens-de-stock-royalty-free-neur%C3%B4nios-image6909719, acessado em novembro de 2012.

[3] ADOBE SYSTEMS. **Adobe Flash**. Disponível em http://www.adobe.com/products/flash.html, acessado em outubro de 2012.

[4] AHN, C. W.; RAMAKRISHNA, R. S. **A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations**. *IEEE Transactions on Evolutionary Computation*, 6(6), 2002.

[5] AIRES, R. R. S. **Visual Analysis of Regulatory Networks**. IST, disponível em https://dspace.ist.utl.pt/bitstream/2295/803248/1/dissertacao.pdf, acessado em outubro de 2012.

[6] ALARCÓN, J. A. B. L. **Airetama - Um Arcabouço Baseado em Sistemas Multiagentes para a Implantação de Comunidades Virtuais de Prática na Web**. UFG, disponível em http://bdtd.ufg.br/tedesimplificado/tde_busca/arquivo.php?codArquivo=1669, acessado em setembro de 2012, 2010.

[7] ALARCÓN, J. A. B. L.; CARVALHO, C. L. **Desenvolvimento de Agentes com JADE**. UFG, 2008.

[8] ALEXANDRE, D. S.; TAVARES, J. M. R. S. **Introduction of Human Perception in Visualization**. FEUP, disponível em http://repositorio-aberto.up.pt/handle/10216/25329, acessado em outubro de 2012, 2010.

[9] AMAZON.COM. **The Internet Movie Database**. Disponível em http://www.imdb.com, acessado em agosto de 2012.

[10] ARAÚJO, A. **As pontes de Königsberg**. Universidade de Coimbra, disponível em http://www.mat.uc.pt/~alma/escolas/pontes, acessado em julho de 2012.

[11] ARAUJO, S. A.; LIBRANTZ, A. F.; ALVES, W. A. **Uso de algoritmos genéticos em problemas de roteamento de redes de computadores.** *Exacta*, 5:321–327, 2007.

[12] ARTERO, A. O. **Inteligência Artificial: Teoria e Prática**. Livraria da Física, 2009.

[13] AZEVEDO, T. A.; RODRIGUEZ, M. V. R. **Softwares para Análise de Redes Sociais - ARS**. *VI Congresso Nacional de Excelência em Gestão*, 2010.

[14] BARABASI, A. L. **Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life**. Plume, 2003.

[15] BARABASI, A. L.; ALBERT, R. **Emergence of Scaling in Random Networks**. *SCIENCE*, 286:509, 1999.

[16] BARABASI, A. L.; ALBERT, R.; JEONG, H. **The Diameter of the World Wide Web**. *Nature*, 401:130–131, 1999.

[17] BEARMAN, P.; ET AL. **Chains of Affection: The Structure of Adolescent Romantic and Sexual Networks**. *American Journal of Sociology*, 110(1):44–91, 2004.

[18] BEDERSON, B.; GROSJEAN, J. **Piccolo Toolkit**. Disponível em http://www.cs.umd.edu/hcil/jazz, acessado em novembro de 2012.

[19] BELLIFEMINE, F.; ET AL. **JADE Programmer's Guide**. Disponível em http://jade.tilab.com/doc/programmersguide.pdf, acessado em julho de 2012.

[20] BELLIFEMINE, F.; CAIRE, G.; GREENWOOD, D. **Developing Multi-Agent Systems with JADE**. Wiley, 2007.

[21] BENTOLILA, D. L.; SOUZA, C. R. **Uma Ferramenta para Auxílio na Avaliação de Usabilidade de APIs**. UFPA, disponível em http://http://www.lbd.dcc.ufmg.br/colecoes/semish/2009/007.pdf, acessado em agosto de 2012.

[22] BOYD, D.; ELLISON, N. B. **Social Network Sites: Definition, History, and Scholarship**. *Journal of Computer-Mediated Communication*, 13(1-2), 2007.

[23] BUCHANAN, M. **Nexus: Small Worlds and the Groundbreaking Science of Networks**. W.W. Norton, 2003.

[24] BURT, R. S. **Structural Holes: The Social Structure of Competition**. Harvard University Press, 1995.

[25] CAMPOS, V. B. G. **Algoritmos para Resolução de Problemas em Redes**. IME, disponível em http://aquarius.ime.eb.br/~webde2/prof/vania/apostilas/Apostila-Redes.pdf, acessado em outubro de 2012.

[26] CARD, S.; MACKINLAY, J.; SHNEIDERMAN, B. **Readings in Information Visualization: Using Vision to Think**. Morgan Kaufmann, 1999.

[27] CARD, S.; MORAN, T.; NEWELL, A. **The Psychology of Human-Computer Interaction**. CRC PressINC, 1983.

[28] CASTRO, L. N. **Inteligência de Enxame**. Unicamp, disponível em ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia013_1s07/topico4_07.pdf, acessado em outubro de 2012.

[29] CHITRA, C.; SUBBARAJ, P. **A Nondominated Sorting Genetic Algorithm for Shortest Path Routing Problem**. *International Journal of Electrical and Computer Engineering*, (5:1), 2010.

[30] CHRISTAKIS, N. A.; FOWLER, J. H. **Connected: The Surprising Power of Our Social Networks and How They Shape Our Lives**. Little, Brown and Co., 2009.

[31] COMUNIDADE BRASILEIRA DE POSTGRESQL. **PostgreSQLBR**. Disponível em http://www.postgresql.org.br, acessado em novembro de 2012.

[32] CORMEN, T.; ET AL. **Introduction to Algorithms**. The MIT Press, 2ª edition, 2001.

[33] COUTO, T. B.; SILVA FILHO, A. M. **Aplicação web usando arquitetura MVC**. *Revista Engenharia de Software Magazine*, (48), 2012.

[34] CUNHA, T. **A Control for Graph Representation and Interaction**. FEUP, disponível em http://repositorio-aberto.up.pt/bitstream/10216/57624/2/Texto%20integral.pdf, acessado em novembro de 2012.

[35] DAVIS JR, C. A. **Aumentando a Eficiência da Solução de Problemas de Caminho Mínimo em SIG**. *In: GIS Brasil 97*, 1997.

[36] DENEUBOURG, J.; ET AL. **The Self-Organizing Exploratory Pattern of the Argentine Ant**. *Journal of Insect Behavior*, 3(2):159–168, Mar. 1990.

[37] DIAS, M. P.; CARVALHO, J. O. **A Visualização da Informação e a sua contribuição para a Ciência da Informação**. DataGramaZero. Disponível em http://www.datagramazero.org.br/out07/Art_02.htm, acessado em julho de 2012.

[38] DIJKSTRA, E. **EWD316: A Short Introduction to the Art of Programming**. Technische Hogeschool, 1971.

[39] DODDS, P. S.; MUHAMAD, R.; WATTS, D. J. **An Experimental Study of Search in Global Social Networks**. *Science*, 301(5634):827–829, Aug. 2003.

[40] DORIGO, M.; DI CARO, G. D.; GAMBARDELLA, L. **Ant Colony Optimization: A New Meta-Heuristic**. In: *Proceedings of the Congress on Evolutionary Computation*, volume 2, p. 1470–1477, Washington D.C., USA, Jun-Sep 1999. IEEE Press.

[41] DORIGO, M.; GAMBARDELLA, L. M. **Ant Colonies for the Traveling Salesman Problem**. Universidade Livre de Bruxelas, disponível em http://www.idsia.ch/~luca/acs-bio97.pdf, acessado em novembro de 2012, 1997.

[42] DORIGO, M.; GAMBARDELLA, L. M. **Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem**. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, Apr. 1997.

[43] DORIGO, M.; MANIEZZO, V.; COLORNI, A. **The Ant System: Optimization by a colony of cooperating agents**. *IEEE Transctions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[44] DORSEY, J. **Twitter**. Disponível em https://twitter.com, acessado em outubro de 2012.

[45] EICH, B. **JavaScript**. Disponível em https://developer.mozilla.org/en/JavaScript, acessado em novembro de 2012.

[46] FRANCO, A. **Fluzz: Vida humana e convivência social nos novos mundos altamente conectados do terceiro milênio.** Escola de Redes, disponível em http://pt.scribd.com/doc/48960099/Fluzz-Augusto-de-Franco, acessado em outubro de 2012, 2011.

[47] GEPHI CONSORTIUM. **Gephi - The Open Graph Viz Platform**. Disponível em https://gephi.org, acessado em novembro de 2012.

[48] GOOGLE INC. **YouTube**. Disponível em http://http://www.youtube.com/, acessado em novembro de 2012.

[49] GOSS, S.; ARON, S.; DENEUBOURG, J.; PASTEELS, J. **Self-Organized Shortcuts in the Argentine Ant**. *Naturwissenschaften*, 76(12):579–581, Dec. 1989.

[50] GRANOVETTER, M. S. **The Strength of Weak Ties**. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[51] GRANOVETTER, M. **A Theoretical Agenda for Economic Sociology**. Technical report, Institute for Research on Labor and Employment, UC Berkeley, June 2000.

[52] GRASSÉ, P. **Les Insectes Et Leur Univers**, volume 22 de **Conférences du Palais de la découverte**. Université, 1949.

[53] GRASSE, P. P. **La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs.** *Insectes Sociaux*, 6:41–81, 1959.

[54] GRÉGIO, A. R. A.; SANTOS, R. **Análise e Visualização de Logs de Segurança**, volume 1. São José, 2010.

[55] HE, F.; QI, H.; FAN, Q. **An Evolutionary Algorithm for the Multi-objective Shortest Path Problem.** *Institute of Systems Engineering*.

[56] HEER, J.; BOYD, D. **Vizster: Visualizing Online Social Networks.** *IEEE Information Visualization (InfoVis)*, p. 32–39, 2005.

[57] HEER, J.; BOYD, D. **Vizster**. Disponível em http://hci.stanford.edu/jheer/projects/vizster, acessado em novembro de 2012.

[58] HEER, J.; CARD, S. K.; LANDAY, J. A. **Prefuse**. Disponível em http://prefuse.org, acessado em julho de 2012.

[59] IEEE COMPUTER SOCIETY. **The Foundation for Intelligent Physical Agents (FIPA)**. Disponível em http://www.fipa.org, acessado em novembro de 2012.

[60] JOHNSON, S. **Emergence: The Connected Lives of Ants, Brains, Cities, and Software**. Allen Lane, 2001.

[61] JOHNSON, S. **Where Good Ideas Come From: The Natural History of Innovation**. Riverhead Books, 2010.

[62] KALAMARAS, D. V. **SocNetV**. Disponível em http://socnetv.sourceforge.net/index.html, acessado em novembro de 2012.

[63] KNUTH, D. **Selected Papers on Analysis of Algorithms**. Csli Lecture Notes. Center for the Study of Language and Inf, 2001.

[64] KUMAR, R.; KUMAR, M. **Exploring Genetic Algorithm for Shortest Path Optimization in Data Networks**. *Global Journal of Computer Science and Technology*, 10, 2010.

[65] LAUMANN, E. **The Social Organization of Sexuality: Sexual Practices in the United States**. University of Chicago Press, 1994.

[66] LEMIEUX, V.; OUIMET, M. **L'analyse Structurale des Reseaux Sociaux**. De Boeck, 2004.

[67] LIMA, M. **Visual Complexity**. Université de Montréal, disponível em http://www.visualcomplexity.com/vc, acessado em outubro de 2012.

[68] LIMA, M. **Visual Complexity: Mapping Patterns of Information**. Princeton Architectural Press, 2011.

[69] LOCAWEB. **ASP.Net MVC, o que que é isso?** Disponível em http://blog.locaweb.com.br/tecnologia/aspnet-mvc-o-que-e-isso, acessado em agosto de 2012.

[70] MACEDO, J. A.; VITALI, M. M. **Algoritmo de Dijkstra. Estudo e Implementação**. UFES, disponível em http://claudiaboeres.pbworks.com/f/apresentacao-JoseAlexandre-e-Maycon.pdf, acessado em julho de 2012.

[71] MACKINLAY, J. **Automating the Design of Graphical Presentations of Relational Information**. *ACM Trans. Graph.*, 5(2):110–141, Apr. 1986.

[72] MCLAREN, D. **Mentionmapp**. Disponível em http://mentionmapp.com, acessado em novembro de 2012.

[73] MICROSOFT. **Microsoft Windows**. Disponível em http://windows.microsoft.com, acessado em setembro de 2012.

[74] MILGRAM, S. **The Small World Problem**. Psychology Today, May 1967.

[75] NASCIMENTO, H. A. D.; FERREIRA, C. B. R. **Uma introdução à visualização de informações**, volume 9. Revista Visualidades (UFG), Jul-Dez 2011.

[76] NOKIA. **QT**. Disponível em http://qt.digia.com, acessado em julho de 2012.

[77] OLIPHANT, F. **TagGraph**. Disponível em http://taggraph.com, acessado em julho de 2012.

[78] O'MADADHAIN.; ET AL. **JUNG (Java Universal Network/Graph Framework)**. Disponível em http://jung.sourceforge.net, acessado em novembro de 2012.

[79] O'MADADHAIN, J.; ET AL. **Analysis and Visualization of Network Data Using JUNG**. UCI, disponível em http://www.ics.uci.edu/~smyth/kddpapers/UCI_KD-D_JUNG_preprint.pdf, acessado em agosto de 2012.

[80] O'MADADHAIN, J.; ET AL. **JUNG: A Brief Tour**. Disponível em http://jung.sourceforge.net/presentations/JUNG_M2K.pdf, acessado em julho de 2012.

[81] ORACLE. **JAVA**. Disponível em http://www.java.com, acessado em novembro de 2012.

[82] ORACLE. **MySQL**. Disponível em http://www.mysql.com, acessado em outubro de 2012.

[83] OSGI ALLIANCE. **OSGi technology**. Disponível em http://www.osgi.org/Main/HomePage, acessado em novembro de 2012.

[84] PISARUK, F. **K-menores caminhos**. IME (USP), disponível em http://www.teses.usp.br/teses/disponiveis/45/45134/tde-14072009-185725/publico/mestrado.pdf, acessado em novembro de 2012.

[85] POLTHIER, K.; ET AL. **Java View**. Disponível em http://www.javaview.de, acessado em novembro de 2012.

[86] RAPOPORT, A. **Mathematical Models of Social Interaction**, volume 2 de **Handbook of Mathematical Psychology**. Wiley, 1963.

[87] RECUERO, R. **Redes Sociais na Internet**. Sulina, 2009.

[88] SANTOS, R. **Mineração e Visualização de Dados usando Java**. INPE, disponível em http://www.lac.inpe.br/~rafael.santos/Docs/CTI2010/JavaDMVis.pdf, acessado em outubro de 2012.

[89] SCHARNOW, J.; TINNEFELD, K.; WEGENER, I. **The Analysis of Evolutionary Algorithms on Sorting and Shortest Paths Problems**. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366, 2005.

[90] SIMÕES, A.; COSTA, E. **Inteligência Artificial: Fundamentos e Aplicações**. FCA, 2008.

[91] SINGH, J. **Collaborative Networks as Determinants of Knowledge Diffusion Patterns**. *Management Science*, May 2004.

[92] SIPSER, M. **Introduction to the Theory of Computation**. PWS Pub. Co., 1996.

[93] SOUZA JUNIOR, S. F.; SOUZA, C. R. B. **Visualização Integrada de Múltiplas Métricas de Redes Sociais**. *WIVA*, 2008.

[94] STALLMAN, R. **Free Software Foundation (FSF)**. Disponível em http://http://www.fsf.org, acessado em setembro de 2012.

[95] TEAM, T. J. F. D. **JUNG Manual**. Disponível em http://jung.sourceforge.net/doc/manual.html, acessado em novembro de 2012.

[96] TELECOM ITALIA. **JADE**. Disponível em http://jade.tilab.com, acessado em novembro de 2012.

[97] THE APACHE SOFTWARE FOUNDATION. **Lucene**. Disponível em http://lucene.apache.org/core, acessado em outubro de 2012.

[98] THE ECLIPSE FOUNDATION. **Eclipse**. Disponível em http://www.eclipse.org, acessado em novembro de 2012.

[99] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL**. Disponível em http://www.postgresql.org, acessado em novembro de 2012.

[100] UNIVERSITY OF CALIFORNIA, B. **BSD (Berkeley Software Distribution)**. Disponível em http://www.bsd.org, acessado em outubro de 2012.

[101] VAN SANDE, S. **Facebook Visualiser**. Disponível em http://vansande.org/facebook/visualiser, acessado em novembro de 2012.

[102] VAUCHER, J.; NCHO, A. **JADE Tutorial and Primer**. Université de Montréal, disponível em http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html, acessado em novembro de 2012.

[103] VENTER, J. C.; ET AL. **The Sequence of the Human Genome**. *Science*, 291, 2001.

[104] WARE, C. **Information Visualization: Perception for Design**. The Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann Pub-S, 2004.

[105] WASSON, G.; TJADEN, B. **The Oracle of Kevin Bacon**. Disponível em http://oracleofbacon.org, acessado em setembro de 2012.

[106] WATTS, D. J. **Six Degrees: The Science of a Connected Age**. Norton, 2003.

[107] WATTS, D. J.; DODDS, P.; NEWMAN, M. **Identity and Search in Social Networks**. *Science*, 296:1302–1305, 2002.

[108] WATTS, D. J.; STROGATZ, S. H. **Collective dynamics of 'small-world' networks**. *American Journal of Sociology*, 393(2):440–442, June 1998.

[109] YAHOO! INC. **Flickr**. Disponível em http://www.flickr.com, acessado em outubro de 2012.

[110] YWORKS. **yfiles**. Disponível em http://www.yworks.com/en/products_yfiles_about.html, acessado em novembro de 2012.

[111] ZIVIANI, N. **Projeto de Algoritmos com Implementações em Pascal e C**. Pioneira, 1999.

[112] ZUCKERBERG, M.; ET AL. **Facebook**. Disponível em www.facebook.com, acessado em novembro de 2012.