I-035 – TUTORIAL DA FERRAMENTA TOOLKIT DO EPANET PARA PROGRAMADORES

Luis Henrique Magalhães Costa[(1)]
Engenheiro Civil pela Universidade Federal do Ceará (2003). Doutor em Recursos Hídricos pelo Departamento de Engenharia Hidráulica e Ambiental – UFC e Instituto Superior Técnico de Lisboa (PEDD-CAPES).
Artemisa Fontinele Frota[(2)]
Graduanda em Engenharia Civil pela Universidade Estadual Vale do Acaraú. Bolsista do programa PIBIC/CNPq.

Endereço[(1)]**:** Av. da Universidade, 132 - Betânia - Sobral - CE - CEP: 62100-000 - Brasil - Tel: (88) 3677-4219 - e-mail: luishenrique.uva@gmail.com

RESUMO

O Epanet é um programa de computador que permite a análise dos parâmetros de uma rede de distribuição de água através de simulações estáticas e dinâmicas. O Epanet Programmer's Toolkit é uma DLL (Dynamic Link Library) composta por funções que possibilitam aos programadores interagir diretamente com o Epanet, modificando seu mecanismo computacional de acordo com suas necessidades específicas.
Desse modo, a toolkit do Epanet pode ser utilizada para elaboração de programas com variadas áreas de aplicação, podendo citar como exemplos a otimização de diâmetros de redes de distribuição, a reabilitação e calibração de sistemas de abastecimento de água e a otimização das estratégias de operações de bombas. Tendo em vista que o toolkit é uma ferramenta gratuita e de fácil acesso, com inúmeras utilidades que permitem diminuir custos e até mesmo melhorar a qualidade do abastecimento de água, esse tutorial é um meio de aprendizagem fácil e objetivo para programadores que desejam aprender a utilizar esta poderosa ferramenta.

PALAVRAS-CHAVE: Epanet, Tutorial, Otimização.

INTRODUÇÃO

O acesso à água de qualidade e em suficiente quantidade é fundamental para saúde e desenvolvimento da população. Desse modo, o adequado funcionamento das redes de distribuição de água se destaca como fator importante para a saúde, o bem estar e a economia das massas beneficiadas por elas. No Brasil, quase todas as cidades já possuem rede de abastecimento de água, entretanto, é comum problemas como falta de água em horários de pico, pressão insuficiente em determinados pontos e até mesmo falta de água por prolongados períodos de tempo.

Atualmente, com o desenvolvimento da tecnologia, a utilização de softwares para solucionar problemas de modo rápido, fácil e eficiente é a melhor alternativa para aperfeiçoar o funcionamento das redes de distribuição de água e diminuir os custos com a implantação de novas redes ou com a ampliação das já existentes. Tendo isso em vista, o presente trabalho apresenta o tutorial da toolkit do Epanet e consiste em uma ferramenta bastante útil para pesquisadores que buscam desenvolver aplicativos que automatizem a análise dos sistemas de distribuição de água.

O Epanet é um programa de computador que permite executar simulações estáticas e dinâmicas do comportamento hidráulico e da qualidade da água em redes de distribuição pressurizada, sendo o programa de modelagem hidráulica mais utilizado no mundo devido sua facilidade de uso e fornecimento gratuito. A toolkit do Epanet, por sua vez, é uma DLL (Dynamic Link Library) composta por várias funções que possibilitam ao programador personalizar o mecanismo computacional do simulador hidráulico Epanet de acordo com suas necessidades específicas.

Desse modo, esse tutorial tem como objetivo ensinar a utilizar a toolkit do Epanet de modo que o programador saiba como obter ou modificar parâmetros de uma rede de distribuição de água, assim como executar

simulações estáticas e dinâmicas, utilizando as funções adequadas para cada situação, já que atualmente só é possível encontrar manuais do toolkit em outro idioma e com códigos nas linguagens C, C++, Pascal e Visual Basic, além de apresentarem uma metodologia basicamente expositiva.

Assim, esse trabalho busca não só expor as funções como também explicar as variáveis de entrada e saída de cada função, quais os possíveis erros que podem ocorrer ao utilizar algumas funções e, através das aplicações práticas, mostrar a importância de saber utilizar corretamente a toolkit do Epanet para criar aplicações úteis e que permitam automatizar as análises dos sistemas de abastecimento de água.

MATERIAIS E MÉTODOS

A metodologia utilizada consiste em apresentar as principais funções da toolkit do Epanet, explanando a sintaxe e as aplicações de cada uma delas, através de exemplos de aplicações práticas. A linguagem de programação adotada para elaboração dos algoritmos foi C# e a IDE (Integrated Development Enviroment) utilizada foi o Visual Studio.

A linguagem C# foi escolhida por ser uma linguagem acessível e de fácil entendimento que vem crescendo bastante e está sendo muito procurada no mercado de trabalho. Ela foi desenvolvida pela Microsoft, o que evita problemas de compatibilidade com o ambiente Windows, possui recursos de programação orientada a objetos e foi desenvolvida buscando corrigir problemas encontrados em outras linguagens.

De início, será mostrado um algoritmo do tipo aplicativo de console *(.NET Framework*) a fim de ensinar as funções comuns à maioria dos programas que utilizam a toolkit do Epanet.

O arquivo DLL da toolkit do Epanet é denominado de EPANET2.DLL e é adicionado ao programa através da classe Epanet que está contida no arquivo EpanetCSharpLibrary.cs, com download disponível no site http://www.water-simulation.com/wsp/2013/04/21/using-epanet-toolkit-in-csharp/. Ao adicionar o arquivo ao projeto, é importante lembrar de alterar o *namespace* do arquivo para o mesmo nome do projeto, para que assim a classe Epanet possa ser utilizada como parte da aplicação.

A primeira função a ser utilizada é a função ENopen, que abre a toolkit do Epanet para análise hidráulica de uma determinada rede de distribuição. Segue a sintaxe da função ENopen

int ENopen(string input_file, string report_file, string output_file);

onde o primeiro parâmetro recebe como argumento o nome do arquivo de extensão ".INP", que é gerado através da exportação da rede pelo simulador Epanet, que contém a rede a ser analisada. O segundo parâmetro recebe o nome do arquivo de relatório com a extensão ".RPT" e é usado para registrar quaisquer mensagens de erro que podem ocorrer durante o processamento do arquivo de entrada e as mensagens de status durante a simulação hidráulica. O terceiro parâmetro recebe o nome de um arquivo opcional de saída binário, caso não haja necessidade de salvar o arquivo de saída binário, pode ser utilizada uma *string* vazia como o terceiro parâmetro.

É importante ressaltar que o arquivo contendo a rede deve estar salvo na mesma pasta da aplicativo executável gerado pelo Visual Studio.

Em conjunto com a função ENopen, deve ser utilizada a função ENclose. Esta deve ser empregada sempre que a análise da rede for finalizada, o que ocorre geralmente no final do algoritmo. A utilização dessa função implica no fechamento do sistema da toolkit, incluindo todos os arquivos que estavam sendo processados. Essa função não possui parâmetros e sua sintaxe é apresentada a seguir

int ENclose();

Para a representação via linguagem de programação das partes construtivas de uma rede de distribuição, serão criadas classes para os principais componentes da rede e uma classe que represente a rede como um todo.

Desse modo, a primeira aplicação desenvolvida possuirá apenas as classes No, Trecho e Rede, além da classe Epanet, que é uma classe estática contendo as funções da toolkit responsáveis por vincular o programa com o simulador Epanet. Cada classe deve ter variáveis compatíveis com os parâmetros do determinado componente que se busca representar. Por exemplo, a variável comprimento deve ser declarada na classe Trecho e ser do tipo float, pois no Epanet esse parâmetro pertence aos trechos e pode ser ou não um número inteiro. O fluxograma ilustrado pela Figura 1 explica como estarão distribuídas as variáveis dentro de cada classe.

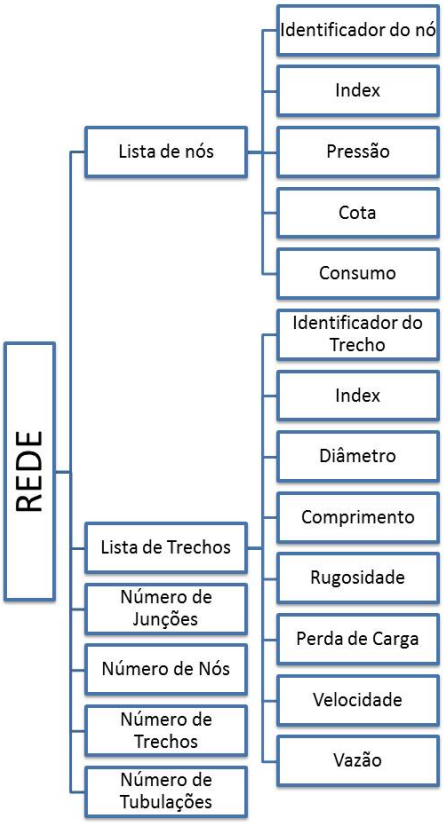

Figura 1: Representação da disposição das variáveis dentro das classes.

Desse modo, para essa aplicação a classe No deve possuirá as variáveis do tipo float cota, pressão e consumo, do tipo inteiro index e do tipo *StringBuilder* id com 31 caracteres, tamanho máximo de caracteres aceitado pelo Epanet para o parâmetro identificador de nó. A Figura 2 apresenta o código da classe No.

```
1  using System.Text;
2
3  namespace exemplo1
4  {
5      class No
6      {
7          public float cota, pressao, consumo;
8          public StringBuilder id = new StringBuilder(31);
9          public int index;
10     }
11 }
```

Figura 2: Código da classe No.

Devido ao fato de a classe do Epanet ter sido convertida de outras linguagens para a linguagem C#, houve problemas de compatibilidade entre a DLL e classe Epanet quando utilizadas as funções que têm como parâmetro a variável id. Para resolvê-los, foi modificado na classe Epanet o tipo da variável id para

StringBuilder. Desse modo, todas as variáveis identificadoras de nó deixaram de ser do tipo *string* para ser do tipo *StringBuilder*.

A classe Trecho possuirá as variáveis do tipo float comprimento, rugosidade, pc (perda de carga), diâmetro, velocidade, vazão, do tipo inteiro index e do tipo StringBuilder id com 31 caracteres. A Figura 3 apresenta o código da classe Trecho.

```
1  using System.Text;
2
3  namespace exemplo1
4  {
5      class Trecho
6      {
7          public float comprimento, rugosidade, diametro, pc, velocidade, vazao;
8          public StringBuilder id = new StringBuilder(31);
9          public int index;
10     }
11 }
```

Figura 3: Código da classe Trecho.

A classe Rede, por sua vez, terá duas listas, uma para os nós e outra para os trechos, e variáveis contadoras do tipo inteiro para junções, nós, trechos e tubulações, sendo elas respectivamente nJuncao, nNos, nTrechos e nTub. A criação de variáveis diferentes para a contagem de nós e junções se justifica pelo fato de que nem toda junção é um nó. Da mesma forma, é necessária a criação de uma variável para contagem de trechos e outra para contagem de tubulações, pois nem todo trecho é uma tubulação. A Figura 4 ilustra o código da declaração de variáveis da classe Rede.

```
1  using System.Collections.Generic;
2  using System.Text;
3  using System;
4
5  namespace exemplo1
6  {
7      class Rede
8      {
9          public List<Trecho> trechos = new List<Trecho>();
10         public List<No> nos = new List<No>();
11         public int nNos=0, nTub=0, nJuncao, nTrechos;
```

Figura 4: Declaração de variáveis da classe Rede.

Ainda na classe Rede, será criada uma função para fazer a leitura de dados da rede. Para isso, serão necessárias as funções ENgetcount, ENsolveH, ENgetnodetype, ENgetnodeid, ENgetnodevalue, ENgetlinktype, ENgetlinkid e ENgetlinkvalue.

A função ENgetcount retorna o número de componentes da rede para um tipo especifícado. Sua sintaxe é a seguinte

int ENgetcount(int codigo_componente, ref int numero_componente);

onde o primeiro parâmetro deve receber um número inteiro que representa o código do componente que se deseja quantificar e o o segundo parâmetro recebe como argumento a variável que irá receber o número de componentes do tipo especificado. A Tabela 1 apresenta o código de cada tipo de componente da rede.

Tabela 1: Códigos para cada tipo de componente da rede.

CÓDIGO	COMPONENTE
0	Junções
1	Reservatórios (Nível Fixo e Variável)
2	Trechos
3	Padrões de Tempo
4	Curvas
5	Controles Simples

A função ENsolveH executa a simulação hidraúlica completa para todos os períodos de tempo. Seu uso é indispensável pois só é possível obter os parâmetros de saída do Epanet, como pressão nos nós e perda de carga nos trechos, após a execução da simulação. Essa função não possue nenhum parâmetro e sua sintaxe é a seguinte

int ENsolveH();

A função ENgetnodetype retorna o tipo de uma determinada junção, especificada através do seu index. Sua sintaxe é mostrada a seguir

int ENgetnodetype(int index, ref int tipo_juncao);

onde o primeiro parâmetro recebe como argumento o index da junção a qual se deseja saber o tipo e o segundo parâmetro rebece a variável que irá receber o código do tipo de junção. Os códigos dos tipos de junções são apresentados na Tabela 2.

Tabela 2: Códigos dos tipos de junções.

CÓDIGO	TIPO DE JUNÇÃO
0	Nó
1	Reservatório de Nível Fixo (RNF)
2	Reservatório de Nível Variado (RNV)

A função ENgetnodeid retorna o identificador da junção especificada e possui a seguinte sintaxe

int ENgetnodeid(int index, StringBuilder id);

onde o primeiro parâmetro recebe como argumento o index da junção que se deseja saber o identificador e o segundo parâmetro a variável que armazenará os caracteres do identificador.

Já a função ENgetnodevalue retorna o valor de um determinado parâmetro de uma junção especificada. O parâmetro será indicado através de um código e a junção através do index. A sintaxe da função é mostrada a seguir

int ENgetnodevalue(int index, int codigo_parametro, ref float valor_parametro);

onde o primeiro parâmetro recebe como argumento o index da junção, o segundo recebe o código do parâmetro da junção que se deseja saber o valor e o terceiro recebe o valor do parâmetro da junção. Os códigos dos parâmetros das junções são mostrados abaixo na Tabela 3, sendo os códigos do número 14 ao 23 aplicados apenas para junções do tipo reservatório de nível variado.

Tabela 3: Códigos dos parâmetros das junções.

CÓDIGO	PARÂMETRO DA JUNÇÃO
0	Cota
1	Demanda base
2	Index do padrão de demanda
3	Coeficiente do emissor
4	Qualidade inicial
5	Qualidade da fonte
6	Index do padrão da fonte
7	Tipo de fonte
8	Nível inicial da água no reservatório de nível variado
9	Demanda atual
10	Carga hidráulica
11	Pressão
12	Qualidade atual
13	Vazão mássica por minuto de uma fonte química
14	Volume inicial de água
15	Código do modelo de mistura
16	Volume da zona de entrada/saída em um reservatório de nível variado de dois compartimentos
17	Diâmetro do reservatório de nível variado
18	Volume mínimo de água
19	Index da curva de volume x profundidade
20	Nível máximo de água
21	Nível mínimo de água
22	Fração do volume total ocupada pela zona de entrada/saída de um RNV de dois compartimentos
23	Coeficiente da taxa de reação em massa

Semelhante a função ENgetnodetype porém aplicada para trechos, a função ENgetlinktype retorna o tipo de um determinado trecho, especificado através do seu index. Sua sintaxe é mostrada a seguir

int ENgetlinktype(int index, ref int tipo_trecho);

onde o primeiro parâmetro recebe como argumento o index do trecho o qual se deseja saber o tipo e o segundo parâmetro rebece a variável que irá receber o código do tipo de trecho. Os códigos dos tipos de trechos são apresentados na a seguir na Tabela 4.

Tabela 4: Códigos dos tipos de trechos.

CÓDIGO	TIPO DE TRECHO
0	Tubulação com válvula de retenção
1	Tubulação
2	Bomba
3	Válvula redutora de pressão
4	Válvula sustentadora de pressão
5	Válvula de perda de carga fixa
6	Válvula reguladora de vazão
7	Válvula de controle de perda de carga
8	Válvula genérica

Similar a função ENgetnode id porém aplicada para trechos, a função ENgetlinkid retorna o identificador do trecho especificado e possui a seguinte sintaxe

int ENgetlinkid(int index, StringBuilder id);

onde o primeiro parâmetro recebe como argumento o index do trecho que se deseja saber o identificador e o segundo parâmetro a variável que armazenará os caracteres do identificador.

Semelhante a função ENgetnodevalue porém aplicada para trechos, a função ENgetlinkvalue retorna o valor de um determinado parâmetro de um trecho especificado. O parâmetro será indicado através de um código e o trecho através do index. A sintaxe da função é mostrada a seguir

int ENgetlinkvalue(int index, int codigo_parametro, ref float valor_parametro);

onde o primeiro parâmetro recebe como argumento o index do trecho, o segundo recebe o código do parâmetro do trecho que se deseja saber o valor e o terceiro recebe o valor do parâmetro do trecho. Os códigos dos parâmetros das junções são mostrados abaixo na Tabela 5.

Tabela 5: Códigos dos parâmetos dos trechos.

CÓDIGO	PARÂMETRO DO TRECHO
0	Diâmetro
1	Comprimento
2	Coeficiente de rugosidade
3	Coeficiente de perda de carga localizada
4	Estado inicial do trecho (0 = fechado, 1 = aberto)
5	Rugosidade para tubulações, velocidade inicial para bombas e configuração inicial para válvulas
6	Coeficiente de reação em massa
7	Coeficiente de reação da parede
8	Vazão
9	Velocidade
10	Perda de carga
11	Estado atual do trecho (0 = fechado, 1 = aberto)
12	Rugosidade para tubulações, velocidade atual para bombas e configuração atual para válvulas
13	Energia gasta em KW

LEITURA DOS PARÂMETROS DA REDE

Na classe principal do programa, será aberta a simulação hidráulica com a função ENopen e criado um objeto do tipo Rede que receberá todas as características da rede contida no arquivo de entrada gerado pelo Epanet por meio da função “Carregar_Rede”. Depois de feita a leitura de dados, a simulação será encerrada utilizando a função ENclose. A Figura 5 ilustra como ficou o código da classe principal do programa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            Epanet.ENopen("input.inp", "report.rpt", "");
            Rede rede = new Rede();
            rede.Carregar_Rede();
            Epanet.ENclose();
        }
    }
}
```

Figura 5: Código da classe principal.

A função “Carregar_Rede” da classe Rede iniciará com a função ENsolveH, para que todos os parâmetros de saída do Epanet sejam calculados. Logo após, será obtida da quantidade de junções da rede através da função ENgetcount. Sabendo a quantidade de junções é possível analisar todas as junções com o auxilio da estrutura de repetição *for*, variando o index da junção, visto que os indexes são números inteiros consecutivos começando do 1.

Para cada junção da rede será armazenada na variável tipoNo o código do tipo de junção, adquirido pela função ENgetnodetype. Caso seu valor seja 0 a junção será do tipo nó, assim, a variável contadora de nós (nNos) terá uma unidade somada ao seu valor, o id do nó será obtido pela função ENgetnodeid, os parâmetros do nó serão obtidos pela função ENgetnodevalue e por fim o nó será adicionado a lista de nós. A Figura 6 ilustra essa parte do algoritmo.

```
public void Carregar_Rede()
{
    Epanet.ENsolveH();

    Epanet.ENgetcount(0, ref nJuncao);
    int tipoNo=0;
    No no;
    for(int i=1; i<=nJuncao;i++)
    {
        Epanet.ENgetnodetype(i, ref tipoNo);
        if(tipoNo==0)
        {
            nNos++;
            no = new No();
            Epanet.ENgetnodeid(i, no.id);
            Epanet.ENgetlinkvalue(i, 0, ref no.cota);
            Epanet.ENgetnodevalue(i, 0, ref no.cota);
            Epanet.ENgetnodevalue(i, 11, ref no.pressao);
            Epanet.ENgetnodevalue(i, 1, ref no.consumo);
            nos.Add(no);
        }
    }
```

Figura 6: Leitura de dados dos nós da rede.

Analogamente a leitura de dados dos nós, será feita a leitura de dados das tubulações. Desse modo, será utilizado a estrutura de repetição *for*, variando o index do trecho e verificando o tipo de trecho por meio da função ENgetlinktype. Caso o trecho seja do tipo 1, ou seja, do tipo tubulação, a variável nTub terá seu valor

aumentado em uma unidade, o id da tubulação será obtido com a função ENgetlinkid, os parâmetros da tubulação serão obtidos com a função ENgetlinkvalue e o trecho será adicionado a lista de trechos.

Finalmente, os dados retirados da rede serão mostrados na tela utilizando um *foreach* para mostrar os dados contidos na lista de nós e outro para mostrar os dados contidos na lista de trechos. A Figura 7 apresenta a parte do algoritmo que contem a leitura dos dados dos trechos e os amostragem dos dados da rede.

```
35              Epanet.ENgetcount(2, ref nTrechos);
36              int tipoTrecho = 0;
37              Trecho trecho;
38              for(int i=1; i<=nTrechos;i++)
39              {
40                  Epanet.ENgetlinktype(i, ref tipoTrecho);
41                  if(tipoTrecho==1)
42                  {
43                      nTub++;
44                      trecho = new Trecho();
45                      Epanet.ENgetlinkid(i, trecho.id);
46                      Epanet.ENgetlinkvalue(i, 0, ref trecho.diametro);
47                      Epanet.ENgetlinkvalue(i, 1, ref trecho.comprimento);
48                      Epanet.ENgetlinkvalue(i, 8, ref trecho.vazao);
49                      Epanet.ENgetlinkvalue(i, 10, ref trecho.pc);
50                      Epanet.ENgetlinkvalue(i, 2, ref trecho.rugosidade);
51                      Epanet.ENgetlinkvalue(i, 9, ref trecho.velocidade);
52                      trechos.Add(trecho);
53                  }
54              }
55
56              Console.WriteLine("-------------Nós-------------");
57              Console.WriteLine("ID  Cota  Pressão  Consumo");
58              foreach(No aux in nos)
59              {
60                  Console.WriteLine("{0}  {1}  {2}  {3}", aux.id, aux.cota,
                      aux.pressao, aux.consumo);
61              }
62              Console.WriteLine("-------------Trecho--------------");
63              Console.WriteLine("ID  Diâmetro Comprimento Rugosidade Vazão
                  Velocidade Perda de Carga");
64              foreach (Trecho aux in trechos)
65              {
66                  Console.WriteLine("{0}    {1}        {2}       {3}      {4}
                      {5}     {6}",aux.id, aux.diametro, aux.comprimento,
                      aux.rugosidade, aux.vazao, aux.velocidade, aux.pc);
67              }
68              Console.ReadKey();
69          }
70      }
71  }
```

Figura 7: Código da leitura de dados dos trechos e amostragem dos dados da rede.

Visto as funções básicas de leitura de dados, serão desenvolvidos três algoritmos com aplicações práticas e finalidades diferentes. Estes abordarão as funções já explicadas e outras funções especificas para cada aplicação, sendo elas necessárias para modificação de parâmetros da rede e execução de simulações dinâmicas.

APLICAÇÃO 1: OTIMIZAÇÃO DE DIÂMETROS DA REDE

Já apresentadas as funções necessárias para leitura do arquivo gerado pelo Epanet e obtenção de dados da rede, esse algoritmo foi desenvolvido para ensinar a utilizar as funções que modificam os parâmetros da rede, de acordo com a necessidade do programador, até que se obtenha o resultado esperado. Esse algoritmo tem como finalidade encontrar a solução de dimensionamento de uma rede de distribuição de água de menor custo, a

partir de uma quantidade de iterações informada pelo usuário, sendo que para cada iteração é gerado um conjunto de diâmetros aleatoriamente.

A melhor solução encontrada é aquela que retorne o menor valor para a função objetivo, representada pela Equação 1, onde NT é o número de trechos da rede, L o comprimento de cada trecho e Cm o custo do metro de tubulação de acordo com o diâmetro.

$$C = \sum_{i=1}^{NT} L_i \times Cm_i \qquad \text{equação (1)}$$

A rede malhada composta por dois anéis, apresentada em Alperovits e Shamir (1977), ilustrada na Figura 8, foi adotada como estudo de caso para esse algoritmo e sua única restrição hidráulica é que as pressões nos nós têm que ser no mínimo 30 mca. A Tabela 6 apresenta os 14 diâmetros possíveis para cada tubulação da rede e seus respectivos custos unitários.

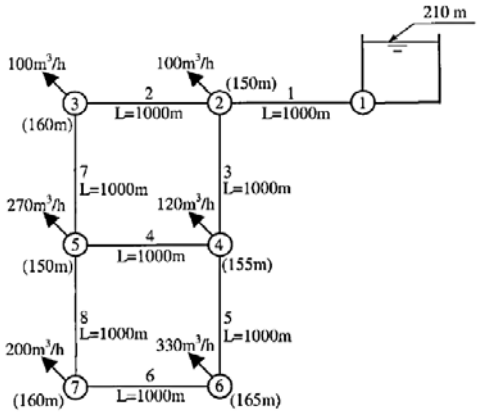

Figura 8: Rede dois anéis de Alperovits e Shamir.

Tabela 6: Custos para tubos.

DIÂMETRO (in)	DIÂMETRO (mm)	CUSTO (un.)
1	25,4	2
2	50,8	5
3	76,2	8
4	101,6	11
6	152,4	16
8	203,2	23
10	254,0	32
12	304,8	50
14	355,6	60
16	406,4	90
18	457,2	130
20	508,0	170
22	558,8	300
24	609,6	550

Da mesma forma que o algoritmo de leitura de dados, esse algoritmo também terá as classes No, Trecho e Rede, das quais apenas a última citada possuirá código diferente do algoritmo anterior. A função "Carregar_Rede" será substituída pela função "Gerar_Solucao", que será responsável por modificar os diâmetros da rede e retorna a solução de menor custo dentre as combinação de diâmetros testadas.

A simulação hidráulica será aberta na classe principal do programa, com a função ENopen e criado um objeto do tipo Rede, que por sua vez terá a rotina "Gerar_Solucao" acionada. Depois de feita a busca pela melhor

solução, a simulação será encerrada utilizando a função ENclose. A Figura 9 ilustra como ficou o código da classe principal do programa.

```
1  namespace Aplicacao1
2  {
3      class Program
4      {
5          static void Main(string[] args)
6          {
7              Epanet.ENopen("input.inp", "report.rpt", "");
8              Rede rede = new Rede();
9              rede.Gerar_Solucao();
10             Epanet.ENclose();
11         }
12     }
13 }
```

Figura 9: Classe principal do algoritmo de otimização de diâmetros.

A função “Gerar_Solucao” terá inicio com a função ENgetcount para obter a quantidade de junções e trechos da rede e com a declaração de variáveis utilizadas no código do programa. O passo seguinte é fazer a leitura da quantidade de trechos e dos comprimentos de cada tubulação da rede. A leitura de dados será feita da mesma forma que foi explicada no algoritmo anterior. A Figura 10 ilustra esses procedimentos.

```
1  using System.Collections.Generic;
2  using System.Text;
3  using System;
4  using System.IO;
5
6  namespace Aplicacao1
7  {
8      class Rede
9      {
10         public List<Trecho> trechos = new List<Trecho>();
11         public int nNos=0, nTub=0, nJuncao, nTrechos;
12
13         public void Gerar_Solucao()
14         {
15             Epanet.ENgetcount(2, ref nTrechos);
16             Epanet.ENgetcount(0, ref nJuncao);
17
18             int tipoTrecho=0, tipoNo=0;
19             float aux=0;
20             Trecho trecho;
21             Random aleat = new Random();
22             float[] diametro = new float[] { 24.5F, 50.8F, 76.2F, 101.6F,
                 152.4F, 203.2F, 254F, 304.8F, 355.6F, 406.4F, 457.2F, 508F,
                 558.8F, 609.6F };
23             float[] preco = new float[] { 2, 5, 8, 11, 16, 23, 32, 50, 60, 90,
                 130, 170, 300, 550 };
24             float custo, custo_minimo = 0;
25             int nOpcoes = 0;
26             StreamWriter arquivo = new StreamWriter("solucoes_viaveis.txt");
27             Boolean viavel;
28
29             for (int i=1;i<=nTrechos;i++)
30             {
31                 Epanet.ENgetlinktype(i, ref tipoTrecho);
32                 if (tipoTrecho == 1)
33                 {
34                     nTub++;
35                     trecho = new Trecho();
36                     Epanet.ENgetlinkvalue(i, 1, ref trecho.comprimento);
37                     trechos.Add(trecho);
38                 }
39             }
40             int[] c = new int[nTub];
41             int[] c_minimo = new int[nTub];
```

Figura 10: Declaração de variáveis e leitura de dados dos trechos.

É importante destacar a função de algumas dessas variáveis declaradas, visto que nem todas possuem nomenclatura que indique sua função. Os vetores diametro e preco representam, respectivamente, os possíveis diâmetros para cada tubulação da rede e seu custo por metro de tubulação em ordem crescente de diâmetro. O vetor c, que possui tamanho igual ao número de tubulações da rede, será preenchido a cada iteração aleatoriamente com os índices dos diâmetros que serão substituídos na rede e o vetor c_minimo armazenará a combinação de diâmetros de menor custo. O objeto aleat do tipo Random será utilizado no algoritmo para fornecer os números aleatórios que representam o índice do diâmetro que serão adotados para cada tubulação.

Em seguida, deve ser inserida pelo usuário a quantidade de iteração, valor armazenado pela variável n. Desse modo as iterações serão realizadas por meio de um *for*, que vai de 1 até n. A cada iteração a variável viável deve ser inicializada como verdadeira e deverão ser substituídos os diâmetros das tubulações na rede, operação que será realizada através da função ENsetlinkvalue.

A função ENsetlinkvalue modifica um determinado parâmetro do trecho especificado e sua sintaxe é mostrada a seguir

int ENsetlinkvalue(int index, int codigo_parametro, float valor_parametro);

onde o primeiro parâmetro recebe como argumento o index do trecho, o segundo terá como argumento o código do parâmetro do trecho que se deseja modificar o valor e o terceiro o valor que o parâmetro do trecho irá receber. A Tabela 5 apresenta os códigos para cada parâmetro do trechos. É importante notar que apenas poderão ser modificados os parâmetros que são dados de entrado do Epanet, ou seja, os parâmetros de código 8, 9, 10 e 13, que são calculados pelo Epanet durante a simulação hidráulica, não poderão ser utilizados como argumento da função ENsetlinkvalue.

Após modificados os valores dos diâmetros da rede, a função ENsolveH deve ser utilizada para executar a simulação hidráulica, para que assim possam ser checadas as pressões nos nós. A verificação da única restrição hidráulica dessa rede (pressão nos nós maior que 30 mca) é feita com a simples leitura das pressões, que foi ensinada no algoritmo de leitura de dados. Devido ao fato da pressão nos nós ser necessária apenas para validar a viabilidade da combinação de diâmetro, não havendo precisão de armazenar seus valores, utilizou-se uma variável auxiliar (aux) para receber o valor da pressão de todos os nós, um por vez. Se o valor da variável auxiliar em algum momento for menor que 30 mca, a variável viável se tornará falsa, inviabilizando a combinação. A Figura 11 ilustra a modificação dos diâmetros da rede e a verificação da restrição hidráulica.

```
43              Console.WriteLine("Digite a quantidade de iterações: ");
44              int n = Convert.ToInt32(Console.ReadLine());
45
46              for(int i=1;i<=n;i++)
47              {
48                  viavel = true;
49                  for(int j=1;j<=nTrechos;j++)
50                  {
51                      c[j - 1] = aleat.Next(14);
52                      Epanet.ENgetlinktype(j, ref tipoTrecho);
53                      if (tipoTrecho == 1) Epanet.ENsetlinkvalue(j, 0,diametro[c
                        [j-1]]);
54                  }
55                  Epanet.ENsolveH();
56                  for(int j=1;j<=nJuncao;j++)
57                  {
58                      Epanet.ENgetnodetype(j, ref tipoNo);
59                      if(tipoNo==0)
60                      {
61                          Epanet.ENgetnodevalue(j, 11, ref aux);
62                          if (aux < 30) viavel = false;
63                      }
64                  }
```

Figura 11: Modificação dos diâmetros da rede e verificação da restrição hidráulica.

Para finalizar o código contido dentro do *for* que gera as iterações, o custo da combinação é calculado, caso esta seja viável, e comparado com o menor custo até então. O custo mínimo inicial será o custo da primeira solução viável, quando esta for encontrada o valor de nOpcoes será aumentado em uma unidade e a partir desse momento o custo mínimo só será atualizado se o custo da combinação for o menor que o custo mínimo até então. As combinações viáveis são salvas no arquivo "solucoes_viaveis.txt" representado pelo objeto arquivo do tipo *StreamWriter*.

Após todas as iterações serem concluidas, o programa mostrará na tela os resultados da busca. Caso nenhuma solução seja encontrada o programa mostrará uma mensagem informando. A seguir, a parte final do código da classe Rede é mostrada pela Figura 12.

```
65                  if(viavel==true)
66                  {
67                      custo = 0;
68                      for (int j = 0; j < nTub; j++) custo = custo + preco[c[j]]
                        * trechos[j].comprimento;
69                      foreach (int d in c) arquivo.Write("{0}  ", diametro[d]);
70                      arquivo.WriteLine("Custo : {0}", custo);
71                      if (nOpcoes == 0)
72                      {
73                          nOpcoes = 1;
74                          c_minimo = c;
75                          custo_minimo = custo;
76                      }
77                      if (custo <= custo_minimo)
78                      {
79                          c_minimo = c;
80                          custo_minimo = custo;
81                      }
82                  }
83              }
84              arquivo.Close();
85              if (nOpcoes == 0) Console.WriteLine("Nenhuma opção viável foi
                  encontrada!");
86              if (nOpcoes != 0)
87              {
88                  foreach (int i in c_minimo) Console.Write("{0}  ", diametro
                      [i]);
89                  Console.WriteLine("Custo mínimo: {0}", custo_minimo);
90              }
91              Console.ReadKey();
92          }
93      }
94  }
```

Figura 12: Parte final do algoritmo da classe Rede.

O fluxograma abaixo, representado pela Figura 13, mostra cada etapa do processo de construção do algoritmo.

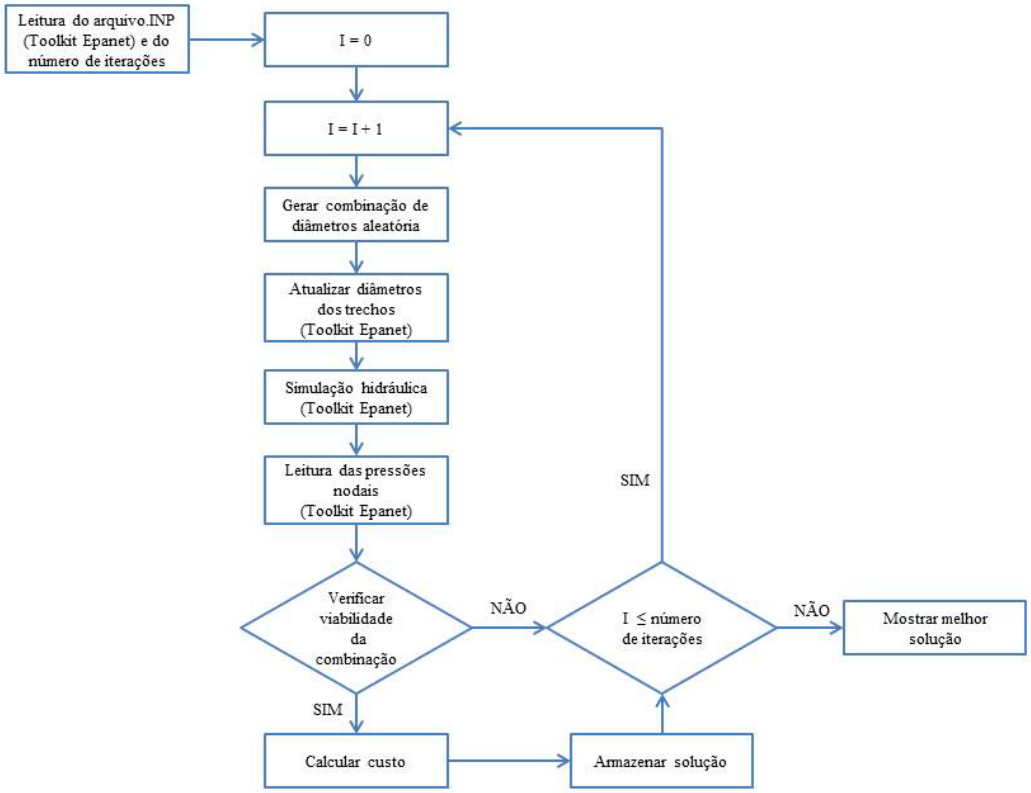

Figura 13: Fluxograma do algoritmo de otimização de diâmetros da rede.

APLICAÇÃO 2: CALIBRAÇÃO DA REDE

Para calibração da rede, ou seja, para que a simulação hidráulica represente com maior precisão possível o comportamento do sistema, esse algoritmo altera a rugosidade das tubulações da rede até que as pressões nodais fiquem próximas de valores pré-estabelecidos, que devem ser obtidos por meio de medições de campo. A calibração também é um processo de otimização que consiste em minimizar o somatório da diferença entre as pressões nodais calculadas e observadas.

Desse modo, o algoritmo de calibração busca encontrar a combinação de rugosidades que minimize o valor da função objetivo, representada pela Equação 2, onde N é a quantidade de nós, P^{obs} a pressão observada e P^{cal} a pressão calculada em cada nó. As rugosidades das tubulações são geradas de forma aleatória e são representadas por uma variável real que corresponde ao coeficiente C de Hazen Williams e varia dentre um intervalo pré-definido.

$$F_{OBJ} = \sum_{i=1}^{N} (P_i^{obs} - P_i^{cal})^2 \qquad \text{equação (2)}$$

A rede de distribuição utilizada para a análise desse algoritmo foi criada por Walski (1983) e adaptada por Gambale (2000) e está representada pela Figura 14, assim como suas características físicas são apresentadas pela Tabela 7. O nível da água no reservatório de nível fixo esta na cota de 60 m e todos os nós da rede estão no mesmo nível, com cota zero. A demanda dos nós e as pressões observadas são detalhadas na Tabela 8.

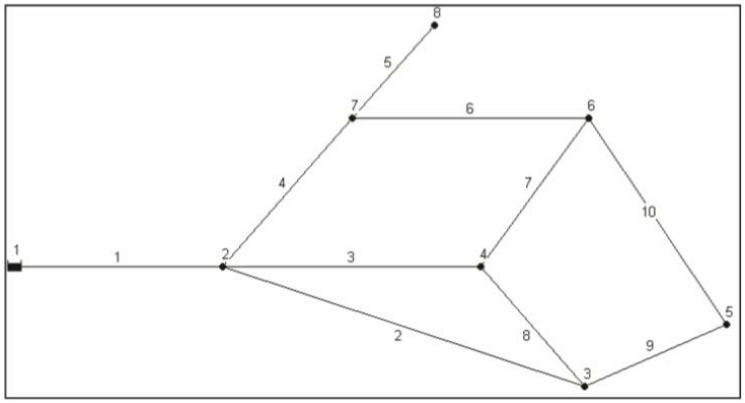

Figura 14: Rede hipotética criada por Walski (1983) e adaptada por Gambale (2000).

Tabela 7: Caracteristicas físicas da rede hipotética.

TUBULAÇÃO	DIÂMETRO (mm)	COMPRIMENTO (m)	RUGOSIDADE C ($m^{0,3698}/s$)
1	500	700	140
2	250	1800	110
3	400	1520	130
4	300	1220	135
5	300	600	90
6	200	1220	110
7	250	920	120
8	150	300	115
9	200	600	85
10	100	1220	80

Tabela 8: Caracteristicas dos nós.

NÓ	DEMANDA (L/s)	PRESSÃO OBSERVADE (mca)
2	0,0	58,74
3	15,0	55,75
4	62,5	56,08
5	15,0	53,77
6	47,5	53,35
7	30,0	54,27
8	37,5	53,03

Nessa aplicação, as classes No, Trecho e principal ficaram praticamente iguais as do programa anterior, alterando apenas o *namespace* do programa para “Aplicacao2” e acrescentando a variável p_observada a classe No, para armazenar o valor das pressões observadas nos nós.

Já a classe Rede terá sua função “Gerar_Solucao” totalmente modificada para se adequar ao problema de calibração da rede. O algoritmo da função iniciará com a declaração das variáveis locais e com a obtenção da quantidade de junções e trechos da rede. Logo após, será feita a leitura da quantidade de tubulações da rede, da mesma forma que foi mostrada anteriormente, e declarados os vetores c e c_minimo, que possuem basicamente as mesmas funções que tinham no algoritmo de otimização de diâmetros, porém, ao invés de armazenarem valores refentes aos diâmetros, apresentaram valores referentes as rugosidades das tubulações. A primeira parte do código da classe Rede é ilustrada na Figura 15.

```
using System.Collections.Generic;
using System.Text;
using System;
using System.IO;

namespace Aplicacao2
{
    class Rede
    {
        public List<Trecho> trechos = new List<Trecho>();
        public List<No> nos = new List<No>();
        public int nNos=0, nTub=0, nJuncao, nTrechos;

        public void Gerar_Solucao()
        {
            int tipoTrecho = 0, tipoNo = 0;
            No no;
            Random aleat = new Random();
            float Fob, Fob_minima = 0;
            StreamWriter arquivo = new StreamWriter("solucoes_viaveis.txt");

            Epanet.ENgetcount(2, ref nTrechos);
            Epanet.ENgetcount(0, ref nJuncao);

            for (int i = 1; i <= nTrechos; i++)
            {
                Epanet.ENgetlinktype(i, ref tipoTrecho);
                if (tipoTrecho == 1) nTub++;
            }
            int[] c = new int[nTub];
            int[] c_minimo = new int[nTub];
```

Figura 15: Declaração de variáveis e contagem de elementos da rede.

O próximo passo será fazer a leitura da quantidade de nós da rede, dos seus respectivos identificadores e indexes. Feita isso, é possível preencher a variável p_observada de cada nó com as pressões observadas de acordo com a Tabela 8, tomando como referência os identificadores de nós. Em seguida, deve ser inserida pelo usuário a quantidade de iteração, valor armazenado pela variável n. Tendo todas as variáveis necessárias para realizar as iterações, uma estrutura de repetição *for* é iniciada, que vai de 1 até n. A Figura 16 ilustra esse procedimento.

```
33              for ( int i=1; i<=nJuncao;i++)
34              {
35                  Epanet.ENgetnodetype(i, ref tipoNo);
36                  if(tipoNo==0)
37                  {
38                      nNos++;
39                      no = new No();
40                      no.index = i;
41                      Epanet.ENgetnodeid(i, no.id);
42                      if (Convert.ToString(no.id) == "2") no.p_observada =
                        58.74F;
43                      if (Convert.ToString(no.id) == "3") no.p_observada =
                        55.75F;
44                      if (Convert.ToString(no.id) == "4") no.p_observada =
                        56.08F;
45                      if (Convert.ToString(no.id) == "5") no.p_observada =
                        53.77F;
46                      if (Convert.ToString(no.id) == "6") no.p_observada =
                        53.35F;
47                      if (Convert.ToString(no.id) == "7") no.p_observada =
                        54.27F;
48                      if (Convert.ToString(no.id) == "8") no.p_observada =
                        553.03F;
49                      nos.Add(no);
50                  }
51              }
52
53              Console.WriteLine("Digite a quantidade de iterações: ");
54              int n = Convert.ToInt32(Console.ReadLine());
```

Figura 16: Leitura de dados dos nós e da quantidade de iterações.

As instruções contidas no laço *for* são ilustradas pela Figura 17.

```
56              for(int i=1;i<=n;i++)
57              {
58                  for(int j=1;j<=nTrechos;j++)
59                  {
60                      c[j - 1] = aleat.Next(80, 140);
61                      Epanet.ENgetlinktype(j, ref tipoTrecho);
62                      if (tipoTrecho == 1) Epanet.ENsetlinkvalue(j, 2,c[j-1]);
63                  }
64                  Epanet.ENsolveH();
65
66                  foreach (No aux in nos) Epanet.ENgetnodevalue(aux.index, 11,
                      ref aux.pressao);
67                  Fob = 0;
68                  foreach (No aux in nos) Fob = Fob + (aux.p_observada -
                      aux.pressao)*(aux.p_observada - aux.pressao);
69
70                  if (i == 1 || Fob <= Fob_minima)
71                  {
72                      Fob_minima = Fob;
73                      c_minimo = c;
74                      foreach (int rugosidade in c) arquivo.Write("{0}  ",
                        rugosidade);
75                      arquivo.WriteLine("Função objetivo = {0}", Fob);
76                  }
77              }
```

Figura 17: Código contido no laço *for* principal.

Como mostra a Figura 16, o código contido no laço *for* principal se inicia preenchendo o vetor c com números aleatórios gerados com o auxílio da variável aleat e que possuem intervalo entre 80 e 140. Utilizando a função ENsetlinkvalue, as rugosidades das tubulações são substituidas pelos números aleatórios armazenados no vetor c e em seguida a simulação hidráulica é executado por meio da função ENsolveH.

Através do estrutura *foreach* é feita a leitura das pressões de cada nó contido na lista nos. Logo após, a variável que representa a função objetivo (Fob) é zerada e seu valor é calculado com o auxílio de outro *foreach*. Caso seja a primeira iteração ou o valor da função objetivo seja menor do que o menor valor de função objetivo encontrado até então, que é representado pela variável Fob_minima, a variável Fob_minima recebe o valor de Fob, o vetor c_minimo será igual ao vetor c e a solução é armazenada em um arquivo.

Feita todas as n iterações, o programa fecha o arquivo de texto e mostra na tela a melhor solução encontrada. A parte final do código é ilustrada na Figura 18.

```
78              arquivo.Close();
79              foreach (int rugosidade in c_minimo) Console.Write("{0}  ",
                  rugosidade);
80              Console.WriteLine("Função objetivo mínima = {0}", Fob_minima);
81              Console.ReadKey();
82          }
83      }
84  }
```

Figura 18: Parte final do código da classe Rede.

O fluxograma abaixo, ilustrado pela Figura 19, mostra cada etapa do processo de construção do algoritmo de modo resumido.

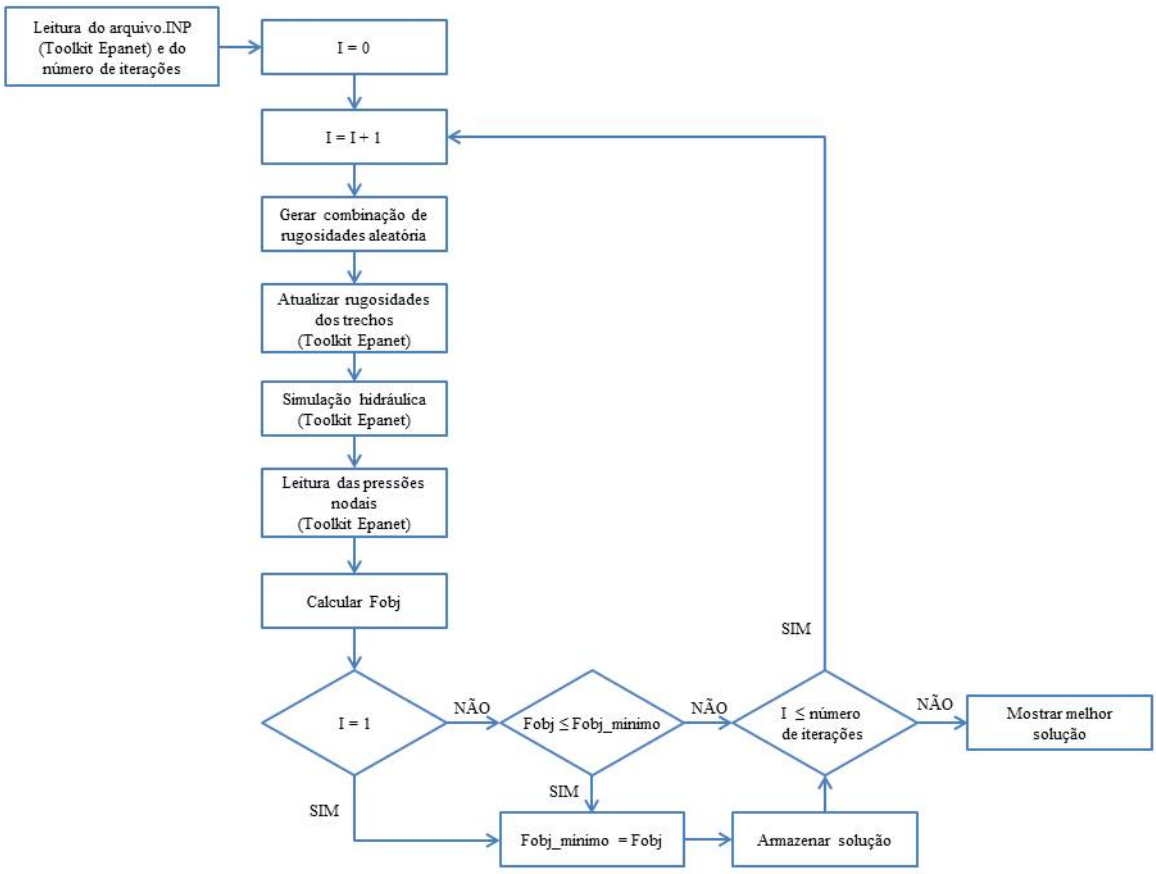

Figura 19: Fluxograma do algoritmo de calibração de redes.

APLICAÇÃO 3: OTIMIZAÇÃO DE ESTRATÉGIA OPERACIONAL

Nesse exemplo é apresentado um algoritmo que, de forma aleatória, gera estratégias operacionais de bombas de um sistema de distribuição e retorna uma solução que forneça um custo energético reduzido e garanta o

abastecimento de água. Desse modo, as estratégias consistem em definir através de uma variável binária o estado de cada bomba, ligada ou desligada, ao longo de um dia.

Para isso, foi utilizada simulação dinâmica, via linguagem de programação. Nesse algoritmo a quantidade de iterações também é informada pelo usuário e a melhor solução encontrada é a estratégia que forneça o menor valor para a função objetivo, representada pela Equação 3, onde N é o número de bombas, T é o número de horas do dia, C é a tarifa de custo de energia, E é o consumo de energia em KWh e X define o estado da bomba, recebendo o valor 1 para ligada e o valor 0 para desligada.

$$F_{OBJ} = \sum_{n=1}^{N} \sum_{t=1}^{T} C_{nt} \times E_{nt} \times X_{nt} \qquad \text{equação (3)}$$

Nesse processo, também é necessário verificar se as soluções são viáveis. Desse modo, a estratégia operacional deve garantir que os níveis dos reservatórios estejam sempre entre os níveis máximo e mínimo e que durante a última hora do dia os níveis dos reservatórios sejam maiores ou iguais aos níveis da primeira hora do dia. A rede escolhida para avaliar o algoritmo de otimização de estratégias de operações foi criada por Francato e Barbosa (1999) e está ilustrada na Figura 20 e possui dois reservatórios de nível vaiável e uma bomba. Os níveis dos reservatórios máximo e mínimo são 9 m e 0 m, respectivamente.

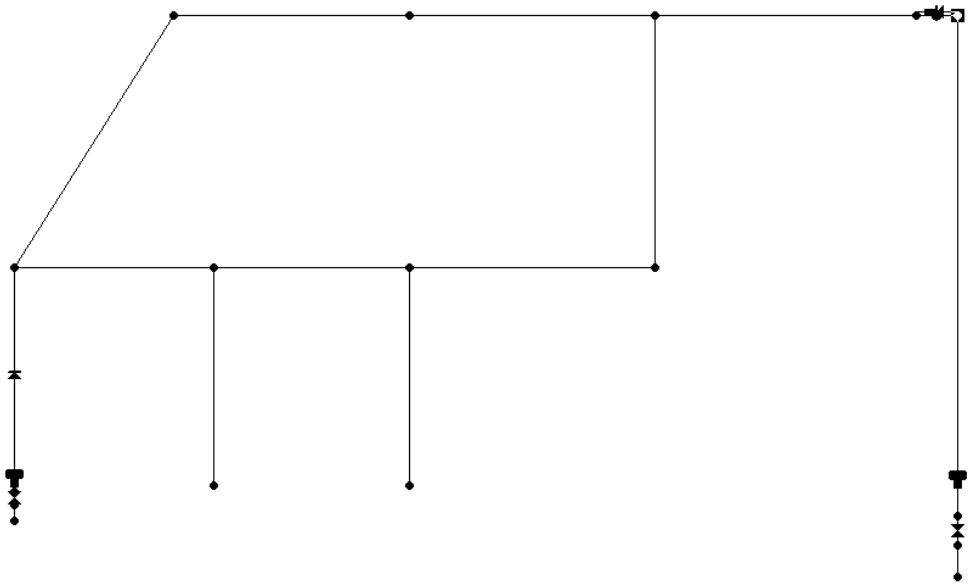

Figura 20: Rede utilizada criada por Francato e Barbosa (1999).

Para essa aplicação prática, o programa precisará de duas novas classes, sendo elas a classe Bomba e a classe RNV. A classe Bomba será dotada de variáveis que caracterizam os parâmetros de uma bomba no Epanet, sendo algumas delas referentes aos padrões associados às bombas. O código da classe Bomba é ilustrado na Figura 21.

Foi utilizado no cógido do algoritmo a regra de que a rede contida no arquivo gerado pelo Epanet deve possuir nomeclatura dos padrões de funcionamento das bombas no formato “p_ + identificador da bomba” e dos padrões de preço no formato “TE_ + identificador da bomba”. As variáveis len_padrao e len_padrao_preco representam, respectivamente, o tamanho dos padrões de funciomento e preço da bomba.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Aplicacao3
{
    public class Bomba
    {
        public List<float> padrao, padrao_preco, vazao, energia;
        public StringBuilder id = new StringBuilder(31);
        public StringBuilder padrao_preco_id = new StringBuilder("TE_", 31);
        public StringBuilder padrao_id = new StringBuilder("p_", 31);
        public float custo;
        public Int32 index, index_padrao, index_padrao_preco, len_padrao,
          len_padrao_preco;

        public Bomba()
        {
            this.padrao = new List<float>();
            this.padrao_preco = new List<float>();
            this.vazao = new List<float>();
            this.energia = new List<float>();
        }
    }
}

```

Figura 21: Código da classe Bomba.

Já a classe RNV possui variáveis que representam parâmetros de um reservatório de nível variado. Seu código está ilustrado na Figura 22.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Aplicacao3
{
    public class RNV
    {
        public float nivel_inicial, nivel_max, nivel_min,nivel_final, cota,
          diametro;
        public StringBuilder id = new StringBuilder(31);
        public List<float> nivel;
        public Int32 index;
    }
}
```

Figura 22: Código da classe RNV.

Como esse algoritmo utilizará simulação dinâmica, será necessário familiarizar com as funções da toolkit que são aplicadas a esse tipo de simulação. Dessa forma, serão descritas essas funções de acordo com a ordem em que serão utilizadas no algoritmo.

A primeira delas é a função ENgettimeparam, que retorna o valor de um determinado parâmetro de tempo em segundos. Essa função possui dois parâmetros, o primeiro é do tipo inteiro e rebece como argumento o código do parâmetro de tempo que se deseja saber o valor. O segundo parâmetro também é do tipo inteiro e recebe como argumento a variável que irá armazenar o valor do parâmetro de tempo. Sua sintaxe é mostrada a seguir

int ENgettimeparam(int codigo_parametro, ref int valor_parametro);

A Tabela 9 apresenta os códigos dos parâmetros de tempo.

Tabela 9: Código dos parâmetros de tempo.

CÓDIGO	PARÂMETRO DE TEMPO
0	Duração da simulação
1	Intervalo de cálculo hidráulico
2	Intervalo de cálculo da qualidade de água
3	Intervalo do padrão de tempo
4	Tempo de início do padrão de tempo
5	Intervalo do tempo de relatório
6	Tempo de início de relatório
7	Intervalo para avaliar os controles baseados em regras
8	Tipo de série temporal de pós-processamento utilizada
9	Número de períodos de relatórios salvos no arquivo binário de saída

Outra função utilizada é a função ENgetpatternindex, que retorna o index de um padrão específicado e possui dois parâmetros. O primeiro deles recebe como argumento o identificador do padrão e o segundo recebe como argumento a variável que irá armazenar o index do padrão de tempo. Sua sintaxe é mostrada a seguir

int ENgetpatternindex(StringBuilder id_padrao, ref int index_padrao);

Já a função ENgetpatternlen é retorna o número de períodos de tempo de um determinado padrão. Também possui dois parâmetros, sendo o primeiro referente ao index do padrão de tempo e o segundo à variável que armazenará o número de períodos de tempo do padrão. Sua sintaxe é mostrada a seguir

int ENgetpatternlen(int index_padrao, ref int numero_periodos);

Outra função importante é a função ENgetpatternvalue, que retorna o fator multiplicador de um período de tempo específico em um padrão de tempo. Essa função possui três argumentos. O primeiro é do tipo inteiro e recebe como argumento o index do padrão de tempo, o segundo recebe como argumento o período de tempo e pode ser entendido como o número do intervalo de tempo que se deseja obter o fator multiplicador e por fim, o terceiro parâmetro, que recebe como argumento a variável que irá armazenar o fator multiplicativo do padrão para aquele determinado período. A sintaxe dessa função é mostrada a seguir

int ENgetpatternvalue(int index_padrao, int periodo, ref float fator_multiplicativo);

Já a função ENopenH é utilizada para abrir o sistema de análise hidráulica e deve ser chamada sempre antes de executar a simulação dinâmica. Essa função não possui argumentos e sua sintaxe é mostrada a seguir

int ENopenH();

Em conjunto com a função ENopenH, deve ser utilizada a função ENcloseH, que fecha o sistema de análise hidráulica, liberando toda a memória alocada. Essa função deve ser chamada sempre após a simulação hidráulica ter terminado. Sua sintaxe é mostrada a seguir

int ENcloseH();

Visto essas funções, o código da classe Rede pode ser iniciado. O código começará com a declaração das variáveis globais da classe, como a lista de reservatórios de nível variado (RNVs), a lista de bombas (bombas), o vetor que armazenará as estratégias (binário) e o vetor que armazenará a melhor estratégia dentre as testadas (melhor_estrategia).

A classe Rede terá sua função “Gerar_Solucao” adaptada para a finalidade de otimização da estratégia operacional da bomba. Ela se inciará com a declaração de variáveis locais e com a leitura da duração da simulação, utilizando a função ENgettimeparam, e da quantidade de junções e trechos da rede, utilizando a função ENgetcount.

A Figura 23 ilustra os processos descritos acima.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.IO;
7
8  namespace Aplicacao3
9  {
10     class Rede
11     {
12         public List<RNV> RNVs = new List<RNV>();
13         public List<Bomba> bombas = new List<Bomba>();
14         Bomba bomba;
15         RNV rnv;
16         public Int32 ntrechos, njuncao, duracao, n_viaveis=0;
17         public long t, tstep;
18         public int tstep_fixo;
19         public float aux, custo_min;
20         public int[] binario = new Int32[24];
21         public int[] melhor_estrategia = new Int32[24];
22
23         public void Gerar_Solucao()
24         {
25             StreamWriter arquivo = new StreamWriter("solucoes_viaveis.txt");
26             Random aleat = new Random();
27             int erro,s,h;
28             int tipoTrecho = 0, tipoNo=0 ;
29             int[] vh = new Int32[25];
30
31             Epanet.ENgettimeparam(0, ref duracao);
32             Epanet.ENgetcount(2, ref ntrechos);
33             Epanet.ENgetcount(0, ref njuncao);
```

Figura 23: Parte inicial do código da classe Rede.

Em seguida será feita a leitura dos parâmetros da bomba. Após lido o identificador da bomba, esses serão acrescentados ao final das variáveis padrao_id e padrao_preco_id. Com os identificadores dos padrão da bomba reconhecidos é possível utilizar a função ENgetpatternindex para obter os indexes dos padrãos.

Logo após, é possível obter os tamanhos dos padrões de tempo utilizando a função ENgetpatternlen. Sabendo a quantidade de intervalos de tempo, ou seja, o tamanho do padrão de tempo, pode-se obter o valor de cada fator multiplicativo para cada período de tempo utilizando a função ENgetpatternvalue.

A Figura 24 mostra o código utilizado para leitura de dados da bomba.

```
35          for (int i = 1; i <= ntrechos; i++)
36          {
37              Epanet.ENgetlinktype(i, ref tipoTrecho);
38              if (tipoTrecho == 2)
39              {
40                  bomba = new Bomba();
41                  bomba.index = i;
42                  Epanet.ENgetlinkid(i, bomba.id);
43                  bomba.padrao_id.Append(bomba.id);
44                  bomba.padrao_preco_id.Append(bomba.id);
45                  Epanet.ENgetpatternindex(bomba.padrao_id, ref
                    bomba.index_padrao);
46                  Epanet.ENgetpatternlen(bomba.index_padrao, ref
                    bomba.len_padrao);
47                  Epanet.ENgetpatternindex(bomba.padrao_preco_id, ref
                    bomba.index_padrao_preco);
48                  Epanet.ENgetpatternlen(bomba.index_padrao_preco, ref
                    bomba.len_padrao_preco);
49                  for(int j = 1; j <= bomba.len_padrao; j++)
50                  {
51                      Epanet.ENgetpatternvalue(bomba.index_padrao_preco, j,
                    ref aux);
52                      bomba.padrao_preco.Add(aux);
53                  }
54              }
55          }
```

Figura 24: Leitura de dados da bomba.

A terceira parte do algoritmo da classe Rede, ilustrada na Figura 25, consiste em fazer a leitura dos níveis máximo e mínimo dos reservatórios de nível variado, do intervalo de tempo da simulação hidráulica (tstep_fixo) e da quantidade de iteração, que será informada pelo usuário. O sistema de análise hidráulica é aberto através da função ENopenH.

```
57          for (int i=1;i<=njuncao;i++)
58          {
59              Epanet.ENgetnodetype(i, ref tipoNo);
60              if (tipoNo == 2)
61              {
62                  rnv = new RNV();
63                  rnv.index = i;
64                  Epanet.ENgetnodevalue(i, 20, ref rnv.nivel_min);
65                  Epanet.ENgetnodevalue(i, 21, ref rnv.nivel_max);
66                  RNVs.Add(rnv);
67              }
68          }
69
70          Epanet.ENgettimeparam(1, ref tstep_fixo);
71
72          Epanet.ENopenH();
73
74          Console.WriteLine("Digite a quantidade de iterações: ");
75          int n = Convert.ToInt32(Console.ReadLine());
```

Figura 25: Terceira parte do algoritmo da classe Rede.

Feita a leitura de todas as variáveis necessária, a próxima parte do algoritmo consiste no *for* principal do algoritmo, que vai da iteração 1 até a iteração n. A cada iteração o padrão de funcionamento da bomba será modificado, a simulação dinâmica será executada e a viabilidade da estratégia será verificada.

O padrão de funcionamento da bomba é modificado através da função ENsetpatternvalue, que modifica o fator multiplicador de um período de tempo específico em um padrão de tempo. Essa função possui três argumentos. O primeiro é do tipo inteiro e recebe como argumento o index do padrão de tempo, o segundo recebe como argumento o período de tempo e pode ser entendido como o número do intervalo de tempo que se deseja

modificar o fator multiplicador e por fim, o terceiro parâmetro, que recebe como argumento valor do fator multiplicativo do padrão para aquele determinado período. A sintaxe dessa função é mostrada a seguir

int ENsetpatternvalue(int index_padrao, int periodo, float valor_fator_multiplicativo);

A variável inteira h representará a hora da simulação e deve ser sempre zerada a cada nova iteração. Da mesma forma, o vetor vh também deve ser zerado. Esse vetor irá armazenar a quantidade de erros que tormam a estratégia operacional para cada hora, que corresponde aos índices dos vetor. Caso a estratégia seja viável, o somatório dos números contidos no vetor vh deve ser igual a zero. A Figura 26 mostra como se inicia o *for* principal da classe Rede.

```
77              for (int i=0;i<n;i++)
78              {
79                  for (int j = 1; j <= bomba.len_padrao; j++)
80                  {
81                      binario[j - 1] = aleat.Next(2);
82                      Epanet.ENsetpatternvalue(bomba.index_padrao, j, binario[j -
                         1]);
83                  }
84                  h = 0;
85                  tstep = tstep_fixo; s=0;
86
87                  for (int j = 0; j < 25; j++) vh[j] = 0;
```

Figura 26: Inicio do *for* principal da classe Rede.

Para a realização da simulação dinâmica serão necessárias três funções: ENinitH, ENrunH e ENnextH. A primeira delas é responsável por inicializar os níveis dos reservatórios de nível variado, os estados e as configurações dos trechos e o tempo do relógio antes de executar a simulação hidráulica. Essa função possui um parâmetro que recebe como argumento os números 0 ou 1, que indicam se os resultados hidraúlicos serão salvos no arquivo hidraúlico. Sua sintaxe é a seguinte

int ENinitH(int parametro_salvar);

Já a função ENrunH é responsável por executar a a simulação hidráulica para um único período e retorna o tempo atual do relógio em segundos. Possui como parâmetro uma variável do tipo *long* que receberá o tempo atual do relógio em segundos. Sua sintaxe é a seguinte

int ENrunH(ref long tempo);

Por último, a função ENnextH, que determina o período de tempo até que o próximo evento hidráulico ocorra. Essa função possui como parâmetro uma variável que armazenará o período de tempo até que o próximo evento hidráulico ocorra e que deve ser tratada apenas como variável de leitura. Caso o valor dessa variável seja zero a simulação dinâmica terá chegado ao fim. Sua sintaxe é a seguinte

int ENnextH(ref long tstep);

Essas três funções devem ser utilizadas em conjunto, sendo necessário a utilização de um *while* que repita as funções ENrunH e ENnextH enquanto a variável retornada pela função ENnextH (tstep) for diferente de zero.

Para cada intervalo de tempo executado dentro do laço while, será verificada cada uma das condições para que a estratégia seja viável. Desse modo, sempre que o resto da divisão do tempo atual do relógio (t) por 3600 for igual a zero e t seja menor que 86400 segundos, a variável h será aumentada em uma unidade. Caso o valor de tstep seja diferente do intervalo de tempo da simulação hidráulica (tstep_fixo) o vetor vh com índice h terá seu valor aumentado em uma unidade. Isso pode ocorrer devido a algum dos reservatório ter atingido o nível máximo ou mínimo durante essa hora.

Caso o tstep seja igual ao tstep_fixo e não seja a primeira hora do simulação, será verificado o nível de todos os reservatórios de nível variado, não podendo estar abaixo do nível mínimo e nem acima do nível máximo. Os níveis inicial e final dos reservatórios são obtidos quando t é igual a zero e 86400, respectivamente. Se o nível inicial for maior que o final, o vetor vh com índice h terá seu valor aumentado em uma unidade.

Por último, será checado se o inteiro retornado pela função ENnextH é diferente de zero, caso este seja, isso significa que houve algum erro na execuçãoe então o vetor vh com índice h terá seu valor aumentado em uma unidade. A Figura 27 mostra a parte final do *for* principal da classe Rede.

```
89              Epanet.ENinitH(0);
90              while (tstep > 0)
91                  {
92                      Epanet.ENrunH(ref t);
93
94                      if (t % 3600 == 0 && t<86400) h++;
95                      if (tstep != tstep_fixo)
96                           vh[h]++;
97                      if(tstep==tstep_fixo && h>1)
98                      {
99                          foreach (RNV rnv in RNVs)
100                         {
101                             Epanet.ENgetnodevalue(rnv.index, 11, ref aux);
102                             if (aux <= rnv.nivel_min || aux >=
    rnv.nivel_max) vh[h]++;
103
104                         }
105                     }
106                     if (t == 0) foreach(RNV rnv in RNVs)
107                         {
108                             Epanet.ENgetnodevalue(rnv.index, 11, ref aux);
109                             rnv.nivel_inicial = aux;
110                         }
111                     if (t == 86400) foreach(RNV rnv in RNVs)
112                         {
113                             Epanet.ENgetnodevalue(rnv.index, 11, ref aux);
114                             if (rnv.nivel_inicial > aux) vh[h]++;
115                         }
116                     erro=Epanet.ENnextH(ref tstep);
117                     if (erro != 0) vh[h]++;
118             }
```

Figura 27: Parte final do *for* principal da classe Rede.

Após terminada a simulação é feita a somatória dos valores contidos no vetor vh (s), caso esta seja igual a zero a estratégia é viável. Para calcular o custo energético é feito basicamente o mesmo procedimento de execução da simulação dinâmica, pegando com a função ENgetlinkvalue a energia gasta pela bomba em KW para cada hora.

Logo após é calculado o custo da estratégia e armazenado no arquivo a estratégia e seu custo. A Figura 28 ilustra os procedimentos realizados para o cálculo do custo da estratégia.

A última parte do algoritmo da classe Rede, mostrada na Figura 29. Para finalizar o for principal da classe, é verificado se a estratégia é a primeira viável a ser encontrada ou se seu custo é menor do que o menor custo até então, se sim o custo mínimo recebe o valor do custo da estratégia e o vetor com a melhor estratégica (melhor_estrategia) recebe o vetor binário.

Após todas as n estratégias aleatórias serem testadas, o sistema de análise hidráulica será fechado através da função ENcloseH e o programa mostrará na tela o número de estratégias viáveis e a melhor estrátegia encontrada.

```
120                 for ( int j = 0; j < 25; j++) s = s + vh[j];
121
122                 if(s==0)
123                 {
124                     n_viaveis = n_viaveis + 1;
125                     bomba.energia.Clear();
126                     tstep = tstep_fixo;
127                     Epanet.ENinitH(0);
128                     while (tstep > 0)
129                     {
130                         Epanet.ENrunH(ref t);
131                         if (t % 3600 == 0 && t < 86400)
132                         {
133                             Epanet.ENgetlinkvalue(bomba.index, 13, ref aux);
134                             bomba.energia.Add(aux);
135                         }
136                         Epanet.ENnextH(ref tstep);
137                     }
138                     bomba.custo = 0;
139                     for (int j = 0; j < 24; j++) bomba.custo = bomba.custo +
                        bomba.energia[j] * bomba.padrao_preco[j];
140
141                     foreach (int estrategia in binario) arquivo.Write("   {0}
                        ", estrategia);
142                     arquivo.Write("Custo: {0}", bomba.custo);
143                     arquivo.WriteLine();
```

Figura 28: Cálculo do custo energético da estratégia.

```
145                     if (n_viaveis == 1 || bomba.custo < custo_min)
146                     {
147                         custo_min = bomba.custo;
148                         melhor_estrategia = binario;
149                     }
150                 }
151             }
152
153             Epanet.ENcloseH();
154             arquivo.Close();
155             Console.WriteLine("Número de opções viáveis: " + n_viaveis);
156             Console.WriteLine("Melhor estratégia: ");
157             foreach(int estrategia in melhor_estrategia) Console.Write("   {0}
                    ", estrategia);
158             Console.WriteLine("Custo mínimo: {0}", custo_min);
159             Console.ReadKey();
160         }
161     }
```

Figura 29: Parte final do algoritmo da classe Rede.

O fluxograma a seguir, ilustrado pela Figura 30, mostra como se realiza o processo de busca pela melhor estratégia de operação de bombas.

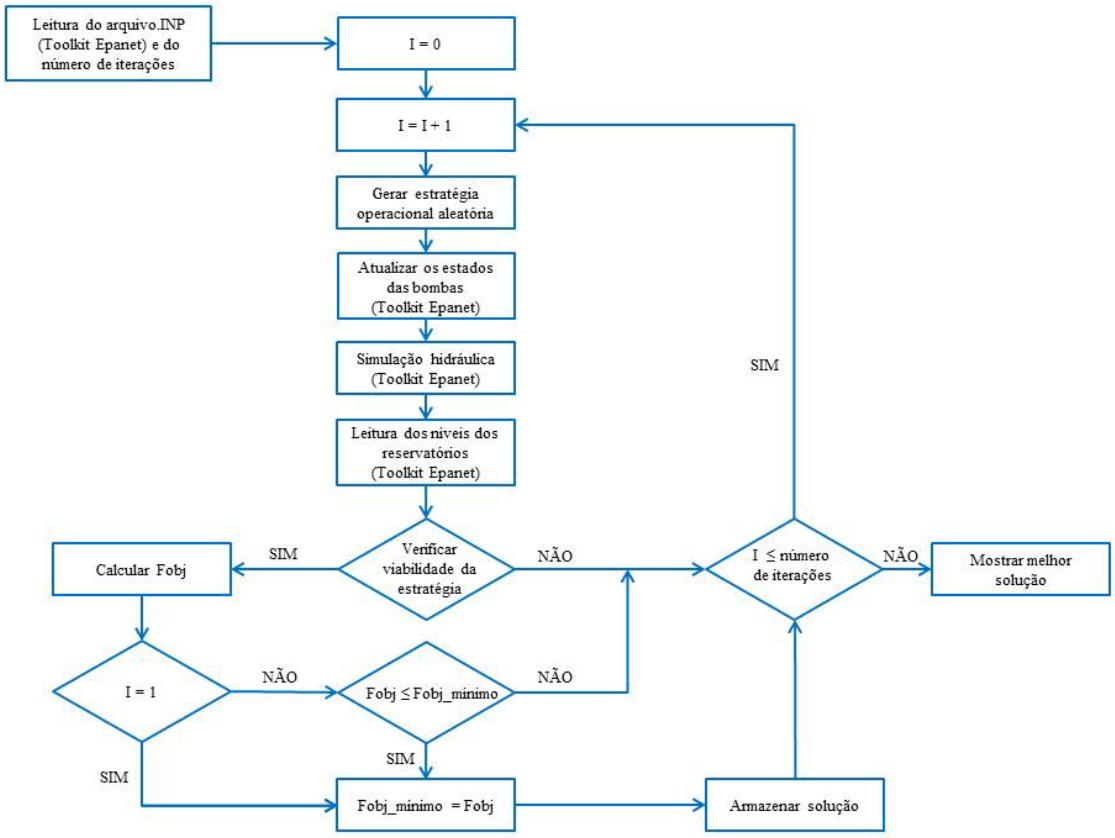

Figura 30: Fluxograma do algoritmo de otimização de estratégias operacionais.

RESULTADOS

Nos algoritmos mostrados nesse trabalho, as variáveis responsáveis por modificar o valor da função objetivo são geradas aleatoriamente e por isso não garantem que a solução encontrada seja a melhor solução possível. Logo, quanto maior for o número de iterações informado pelo usuário maior será a probabilidade de encontrar a melhor solução possível.

Portanto, esse tutorial, além de mostrar aplicações práticas através de algoritmos simples, incentiva a pesquisa na área de análises hidráulicas e proporciona ao programador a facilidade de desenvolver aplicativos robustos com variadas finalidades utilizando a toolkit do Epanet.

CONCLUSÕES

Conclui-se que a toolkit do Epanet é uma ferramenta de grande utilidade para programadores que desejam desenvolver novos aplicativos que possam interagir diretamente com o Epanet. Para isso, esse tutorial representa para esses profissionais uma ferramenta de aprendizagem de fácil compreensão e acesso, pois não só apresenta as principais funções da toolkit do Epanet como explica detalhadamente para que e como utilizá-las.

Vale ressaltar que o tutorial da toolkit do Epanet para programadores é um ótimo recurso para aprender as principais funções do Epanet, podendo o programador posteriormente aprofundar seus conhecimentos sobre a toolkit e desenvolver aplicativos mais robustos capazes de realizar análises importantes na modelagem de sistemas de abastecimento de água.

REFERÊNCIAS BIBLIOGRÁFICAS

1. ALPEROVITS, A., SHAMIR, U. *Design of optimal water distribution systems. Water Resources Research*, v. 13, n. 6, p. 885-900, Dez. 1997.
2. COSTA, L. H. M., PRATA, B. A., RAMOS, H. M., CASTRO, M. A. H. *A branch-and-bound algorithm for optimal pump scheduling in water distribution networks. Water Resources Management*, v. 30, n. 3, p. 1073-1052, Feb. 2016.
3. FRANCATO, A. L., BARBOSA, P. S. F. Operação Ótima de Sistemas Urbanos de Abastecimento de Água. XV Congresso Brasileiro de Engenharia Mecânica. Águas de Lindóia, SP, 1999.
4. GAMBALE, S. R. Aplicação de algoritmo genético na calibração de rede de água. São Paulo, 2000. Dissertação de mestrado em Recursos Hídricos-Escola Politécnica-Universidade de São Paulo, 2000.
5. RAO, Z., SALOMONS, E. *Development of a real-time, near-optimal control system for water distribution networks. Journal of Hydroinformatics*, v. 9, n. 1, p. 25-37, Jan. 2007.
6. ROCHA, V. A. G. M., ARAÚJO, J. K., CASTRO, M. A. H., COSTA, M. G., COSTA, L. H. M. Análise comparative entre RNA, AG e Migha na determinação de rugosidades através de calibração de redes hidráulicas. Revista Brasileira de Recursos Hídricos, v. 18, n. 1, p. 125-134, Jan./Mar. 2013.
7. Rossman, L. A. *Epanet 2: user's manual. National Risk Management Research Laboratory Office of Research and Development. Cincinnati: U.S. Environmental Protection Agency*, 2000.
8. WALSKI, T. M., BRILL, E. D., GESSLER, J., GOULTER, I. C., JEPPSON, R. M., LANSEY, K. E., LEE, H. L., LEIBMAN, J. C., MAYS, L. W., MORGAN, D. R., ORMSBEE, L. E. *Battle of the network models: epilogue. Journal of Water Resources Planning and Management*, v. 113, n. 2, p. 191-203, Mar. 1987.
9. WALSKI, T. M. *Technique for calibration network models. Journal of Water Resources Planning and Management*, v. 109, n. 4, p. 360-372, Out. 1983.