



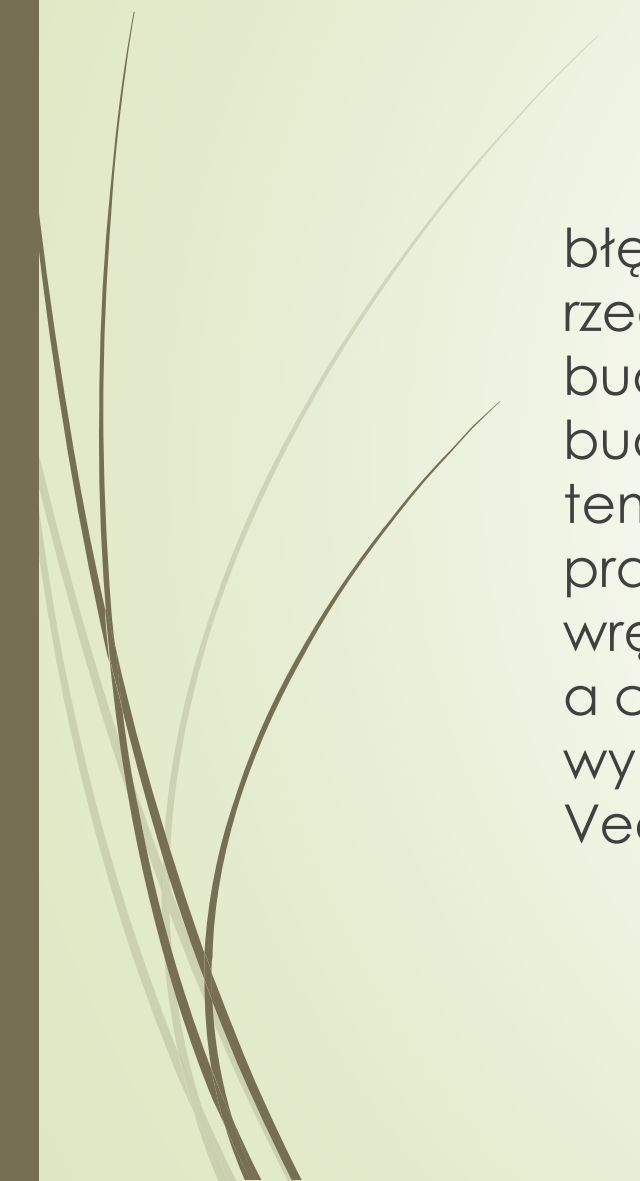
# Pracownia Projektowa

Uzupełnianie braków danych telemetrycznych

Piotr Jastrzębski, Sylwia Nowak, Piotr Romać, Roman Siry,



# Omówienie problemu (1)



Stworzenie odpowiedzi na zadany problem wyszukiwania braków i błędów w danych pochodzących z węzłów ciepłowniczych jest w rzeczywistości problemem trudnym i złożonym czasowo. Na ocieplenie budynku wpływa wiele czynników panujących na zewnątrz badanego budynku takich jak wiatr, słońce, zachmurzenie, wilgotność, temperatura powietrza jak i czynniki wewnętrzne: ilość ludzi, pracujących urządzeń, czajników, kaloryferów, klimatyzacja... Jest wręcz niemożliwym dokładnie przewidzieć temperaturę pomieszczeń a co za tym idzie zużycia ciepła w danym budynku, ponieważ wymagałoby to znajomości danych, które nie są dostępne firmie Veolia.



## Omówienie problemu (2)

Najlepszym rozwiązaniem, uwzględniającym rozróżnienie węzłów cieplowniczych oraz warunków atmosferycznych panujących o różnych porach dnia i roku będzie funkcja wyznaczająca oddzielnie dla każdego węzła cieplnego dane wymagające poprawy ze względu na nierzeczywiste wartości takie jak np. temperatura wody 999,9 °C czy dane potencjalnie wybrakowane oznaczane wartościami 0.00 w bazie danych. Jest to kluczowe założenie rozwiązania naszego problemu.

Ponieważ nie jest znany nam algorytm oraz funkcja która będzie aproksymować z satysfakcjonującą nas dokładnością, musieliśmy wymyśleć odpowiedź osobiście a przyjęte rozwiązanie opierać na zebranych rekordach przedstawionych w plikach csv reprezentujących węzły cieplownicze.



# Ogólny zarys rozwiązania

Oczywistym podejściem jest stworzenie funkcji, która będzie aproksymować wybrane przez nas dane na bazie dostarczonych już danych (sposób aproksymacji będzie zależny od wybranej metody) oraz obudować ją w algorytm, który będzie pomagał w zapełnianiu luk.

Najważniejszym jednak krokiem jest rozróżnienie danych na te, które możemy uznać za całkowicie kompletne (na nich w późniejszym czasie będziemy opierać wszystkie współczynniki potrzebne przy tworzeniu funkcji aproksymującej) oraz takie, które wymagają sprawdzenia poprawności danych oraz ewentualnego poprawienia rekordów z bazy danych.



# Pseudokod zaproponowanego algorytmu

1. Wczytaj dane z plików csv do zaproponowanych struktur
2. Dla każdego węzła wykonaj punkty 3, 4, 5, 6:
3. Podziel dane na poprawne i potencjalnie wymagające poprawy
4. W oparciu o poprawne dane wyznacz współczynniki maksymalnego odchylenia od wartości oczekiwanej zarówno liczbowe jak i procentowe
5. Dla każdej danej potencjalnie wybrakowanej sprawdź poprawność wzorów na ciepło, przepływ, temperaturę zasilającą i powrotną. Jeżeli któryś z parametrów nie mieści się w odchyleniach delta wyznacz go na podstawie funkcji aproksymującej opartej na ważonym wyznaczaniu zaproksymowanej wartości na podstawie roku, miesiąca i dnia
6. Wygeneruj plik csv z poprawnymi danymi nie nadpisując poprzedniego, do pliku tekstowego wstaw raport z algorytmu – założone błędy aproksymacji, poprawione dane, ilość niezmiennych i zmienionych danych.





# Zastosowana technologia

Do rozwiązania problemu stosujemy algorytm napisany w języku C# (.Net 4.0). Dodatkową biblioteką, z której korzystamy jest FileHelpersPPC.dll, która jest biblioteką Open Source ułatwiającą szybkie przeglądanie rekordów z plików csv i rzutowanie ich na wybrany obiekt.

Link do biblioteki : <http://filehelpers.sourceforge.net/>

Wykonany projekt jest typu Console Application i pozwala wyspecyfikować folder, z którego mają być wybierane pliki csv poddawane analizie. Ze względu na jasność kodu i utrzymany styl dobrego programowania istnieje łatwy sposób zastąpienia sczytywania danych z plików csv na podłączenie się do istniejącej już bazy danych poprzez np. Entity Framework i rzutowanie istniejącej tabeli na klasę reprezentującą węzeł ciepłowniczy – Node.

# Klasa Node (1)

```
[DelimitedRecord(";")]
[IgnoreEmptyLines()]
[IgnoreFirst()]
public class Node
{
    private const float MAX_PERCENTAGE_DELTA = 10;
    private const float TEMPERATURE_TZ_MAX = 150f;
    private const float TEMPERATURE_TP_MAX = 100f;
    private const float TEMPERATURE_TZ_MIN = 15f;
    private const float POWER_MAX = 1500;
    private const float FLOW_MAX = 15;
    private const float TOLERANCE = 0.01f;
    private const float cp = 4.1899f;
    public int Location { get; set; }
    public string Address { get; set; }
    public string Counter { get; set; }
    public int Hour { get; set; }
    public int Day { get; set; }
    public int Month { get; set; }
    public int Year { get; set; }
    public float Tz { get; set; }
    public float Tp { get; set; }
    public float PowerkW { get; set; }
    public float FlowM3h { get; set; }
    public float MaxPowerkW { get; set; }
    public float MaxFlowM3h { get; set; }
}
```

# Klasa Node (2)

Zgodnie ze wcześniejszym slajdem klasa Node zawiera właściwości, które są odwzorowaniem kolumn w plikach csv oraz ograniczenia

- MAX\_PERCENTAGE\_DELTA = 10;
- TEMPERATURE\_TZ\_MAX = 150f;
- TEMPERATURE\_TP\_MAX = 100f;
- TEMPERATURE\_TZ\_MIN = 15f;
- POWER\_MAX = 1500;
- FLOW\_MAX = 15;
- TOLERANCE = 0.01f;

Zgodnie z nazewnictwem przyjęto, że wszystkie parametry ( Q, C, Tz, Tp) nie mogą być równe 0 z dokładnością TOLERANCE. Maksymalne dopuszczalne odchylenie danych wyliczonych od oczekiwanych (procentowe) to MAX\_PERCENTAGE\_DELTA a maksymalne Tz, Tp, Q, C są równe odpowiednio 150 °C , 100 °C, 1500kW, 15 m<sup>3</sup>/h. Minimalna temperatura Tz wynosi natomiast 15 °C. Takie ograniczenia przyjęto po wstępnym zapuszczeniu algorytmu wyszukującego maksima każdego z parametrów, średnie oraz mody. Do obliczeń przyjęto cp = 4.1899f, które zostało już sprowadzone do poprawnej jednostki.



## Klasa Node (3)

```
public bool IsNodeComplete()
{
    bool minMax = IsFlowCorrect() && IsTpCorrect() && IsTzCorrect() &&
IsPowerCorrect();
    if (minMax)
    {
        bool percentages = CountPercentageDeltaG() < MAX_PERCENTAGE_DELTA &&
CountPercentageDeltaQ() < MAX_PERCENTAGE_DELTA &&
        CountPercentageDeltaTp() < MAX_PERCENTAGE_DELTA &&
CountPercentageDeltaTz() < MAX_PERCENTAGE_DELTA;
        return percentages;
    }
    return false;
}
```

Metoda IsNodeComplete() sprawdza czy dany rekord może być uznany za poprawny. Pierwszy krok sprawdza mieszczanie się we wcześniej omówionych zakresach, drugi sprawdza czy wszystkie dane są zawarte w granicy błędu 10%. Do tego wykorzystywane są pomocnicze metody omówione poniżej,

# Klasa Node (4)

```
public float CountDeltaQ()  
{  
    return Math.Abs(PowerkW - FlowM3h * cp * (Tz - Tp) * 5f / 18f); // : 3,6  
}  
  
public float CountDeltaG()  
{  
    return Math.Abs(FlowM3h - PowerkW * 3.6f / (cp * (Tz - Tp)));  
}
```

```
public float CountDeltaTp()  
{  
    return Math.Abs(-PowerkW * 3.6f / (FlowM3h * cp) + Tz - Tp);  
}
```

```
public float CountDeltaTz()  
{  
    return Math.Abs(PowerkW * 3.6f / (FlowM3h * cp) + Tp - Tz);  
}
```

```
public float CountPercentageDeltaQ()  
{  
    float expected = FlowM3h * cp * (Tz - Tp) * 5f / 18f;  
    return Math.Abs(PowerkW - expected) / PowerkW; // : 3,6  
}
```

```
public float CountPercentageDeltaG()  
{  
    float expected = PowerkW * 3.6f / (cp * (Tz - Tp));  
    return Math.Abs(FlowM3h - expected) / FlowM3h;  
}
```

```
public float CountPercentageDeltaTp()  
{  
    float expected = -PowerkW * 3.6f / (FlowM3h * cp) + Tz;  
    return Math.Abs(expected - Tp) / Tp;  
}
```

```
public float CountPercentageDeltaTz()  
{  
    float expected = PowerkW * 3.6f / (FlowM3h * cp) + Tp;  
    return Math.Abs(expected - Tz) / Tz;  
}
```

```
public bool IsPowerConnect()  
{  
    return PowerkW < POWER_MAX && PowerkW > TOLERANCE;  
}
```

```
public bool IsFlowCorrect()  
{  
    return FlowM3h < FLOW_MAX && FlowM3h > TOLERANCE;  
}
```

```
public bool IsTzCorrect()  
{  
    return Tz < TEMPERATUR_TZ_MAX && Tz > TEMPERATUR_TZ_MIN && Tz > Tp;  
}
```

# Krok 1 Algorytmu (1)

```
class Program
{
    static void Main(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine("Give path to directory with csv's next time");
            Console.ReadLine();
        }
        Stopwatch sw = new Stopwatch();
        sw.Start();
        Console.WriteLine("Started finding files to fix ...");
        DirectoryInfo d = new DirectoryInfo(args[1]);
        FileInfo[] files = d.GetFiles("*.csv");
        try
        {
            foreach (var file in files)
            {
                Algorithm algorithm = new Algorithm(file);
                algorithm.FixFile();
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        sw.Stop();
        Console.WriteLine("End reading files, time : {0}", sw.Elapsed);
        Console.ReadLine();
    }
}
```



## Krok 1 Algorytmu (2)

W pierwszym etapie dla każdego pliku csv tworzona jest tablica wszystkich rekordów już zrzutowanych na klasę Node. Dane przekazywane są do instancji klasy Algorytm gdzie wykonywane są obliczenia. Dodatkowo Program zawiera wyliczanie długości działania, które pozwala wyznaczać górną barierę np. ilości funkcji aproksymujących w zastosowanym Algorytmie.

# Klasa Algorytm (1)

```
public class Algorithm
{
    private const int FUNCTIONS = 35;
    private FileInfo _file;
    private Node[] _nodes;
    // Lists with Node and int that represents index in array nodes
    private List<Tuple<Node, int>> _completeNodes; // lista potencjalnie dobrych
rekordów
    private List<Tuple<Node, int>> _lackedNodes; // lista potencjalnie
wybrakowanych rekordów
    private float deltaTz;
    private float deltaTp;
    private float deltaQ;
    private float deltaG;
    private float deltaPercentageTz;
    private float deltaPercentageTp;
    private float deltaPercentageQ;
    private float deltaPercentageG;

    public Algorithm(FileInfo fileInfo)
    {
        _file = fileInfo;
        _completeNodes = new List<Tuple<Node, int>>();
        _lackedNodes = new List<Tuple<Node, int>>();
        deltaG = 0.0f;
        deltaQ = 0.0f;
        deltaTp = 0.0f;
        deltaTz = 0.0f;
        deltaPercentageG = 0.0f;
        deltaPercentageQ = 0.0f;
        deltaPercentageTp = 0.0f;
        deltaPercentageTz = 0.0f;
    }
}
```





## Klasa Algorytm (2)

Klasa Algorytm odpowiada w całości za przetworzenie danych i zastąpienie ich poprawnymi. Pole FUNCTIONS odpowiada za ilość funkcji aproksymujących a dokładniej za ich połowę. Na obecnym etapie do szybkiego pokazania Państwu wyników ustalono wartość 35, która w rzeczywistości będzie zwiększona, ponieważ nie mamy czasowych ograniczeń.

Dodatkowo klasa Algorytm posiada kluczowe metody takie jak FlxFile(), FindCoefficients() oraz CountApproximatedValue() omówione na następnym slajdzie.



## Krok 2 Algorytmu

W załączonym kodzie przedstawimy krok po kroku dotychczasowe działanie algorytmu w klasie Algorytm. Załączony kod przewiduje aproksymację na podstawie danych rocznych z wykorzystaniem metod numerycznych aproksymacji funkcjami cosinus i sinus, ponieważ założyliśmy, że występujące wyniki mają charakter cykliczny. Można założyć, że występujące warunki pogodowe jak i wykorzystanie ciepła w ciągu roku jest cykliczne.



# Symulacja

Przedstawimy symulację działania algorytmu dla danych przekazanych z firmy Veolia. Symulacja będzie wykonywać się w czasie możliwie jak najkrótszym, żeby prezentacja nie trwała najdłużej, dlatego wykorzystamy mniej funkcji aproksymujących oraz jedynie aproksymację roczną uwzględniającą co 130 rekord uważany za poprawny dla każdego węzła. Zebrane wyniki przedstawiane są na outpucie konsoli, generowanie raportów do plików tekstowych zostanie wykonane na wersję finalną algorytmu. Do katalogu bin Release lub Debug (w zależności od wybranej wersji odpalania programu) generują się pliki csv zawierające wszystkie rekordy, które były poprawione poprzez algorytm. Pliki nazywane są w konwencji stara\_nazwa\_new.csv.



# Droga do wybranego algorytmu

Przedstawimy teraz krótko wybrane algorytmy, które zostały odrzucone na rzecz wybranej aproksymacji. Krótko omówimy zalety i wady alternatywnych rozwiązań oraz uargumentujemy wybór algorytmu najlepiej aproksymującego dane.



Dziękujemy za uwagę!