Classification dynamique pour le diagnostic de procédés en contexte évolutif

Nathalie Andrea Barbosa Roa

Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Cotutelle internationale avec "Universidad Nacional de Colombia"

Présentée et soutenue par :
Nathalie Andrea Barbosa Roa
le Vendredi 02 décembre 2016

Titre :

A DATA-BASED APPROACH FOR DYNAMIC CLASSIFICATION OF FUNCTIONAL SCENARIOS ORIENTED TO INDUSTRIAL PROCESS PLANTS

École doctorale et discipline ou spécialité :
EDSYS : Automatique 4200046

Unité de recherche :
Laboratoire d'analyse et d'architecture des systèmes (UMR 8001)

Directeur/trice(s) de Thèse :

Louise TRAVE - MASSUYES, Ph. D.
Victor Hugo GRISALES PALACIO, Ph.D.

Jury :
Marie-Véronique LE LANN, Professeur des universités, Président
Rafael GOURIVEAU, Maître de Conférence, Rapporteur
Claudia Victoria ISAZA, Profesor Asociado, Rapporteur
Germán Dário ZAPATA, Profesor Asistente, Examinateur
Louise TRAVE - MASSUYES, Director de Recherche DR1, Directrice de thèse
Victor Hugo GRISALES, Profesor asociado, Directeur de thèse

A DATA-BASED APPROACH FOR DYNAMIC CLASSIFICATION OF FUNCTIONAL SCENARIOS ORIENTED TO INDUSTRIAL PROCESSES

NATHALIE ANDREA BARBOSA ROA

Thesis submitted in order to obtain the title of

DOCTOR

Advisors:
Victor Hugo Grisales Palacio, Ph.D.
Associate Professor

Louise Travé - Massuyès, Ph. D.
Research Director at LAAS-CNRS

UNIVERSIDAD NACIONAL DE COLOMBIA
UNIVERSITÉ TOULOUSE III - PAUL SABATIER
LABORATOIRE D'ANALYSE ET D'ARCHITECTURE DES SYSTÈMES
LAAS–CNRS
TOULOUSE

2016

CLASIFICACIÓN DINÁMICA DE ESCENARIOS FUNCIONALES PARA EL DIAGNÓSTICO DE PROCESOS INDUSTRIALES

NATHALIE ANDREA BARBOSA ROA

300646

Tesis presentada como requisito parcial para obtener el título de

DOCTOR EN INGENIERÍA

PROGRAMA DE DOCTORADO EN INGENIERÍA – INGENIERÍA ELÉCTRICA

Directores:
Victor Hugo Grisales Palacio, Ph.D.
Profesor Asociado

Louise Travé - Massuyès, Ph. D.
Directora de investigación en el LAAS-CNRS

UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
BOGOTÁ D. C.

2016

CLASSIFICATION DYNAMIQUE POUR LE DIAGNOSTIC DE PROCÉDÉS EN CONTEXTE ÉVOLUTIF

NATHALIE ANDREA BARBOSA ROA

Thèse en vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

déliveré par l'Université Toulouse III - Paul Sabatier

Directeurs de thèse:
Louise Travé - Massuyès, Ph. D.
Directeur de Recherche, LAAS-CNRS

Victor Hugo Grisales Palacio, Ph.D.
Professeur Agrégé

UNIVERSITÉ TOULOUSE III - PAUL SABATIER
ÉCOLE DOCTORALE SYSTÈMES (EDSYS) - SYSTÈMES AUTOMATIQUES
LABORATOIRE D'ANALYSE ET D'ARCHITECTURE DES SYSTÈMES
LAAS–CNRS
TOULOUSE
2016

RESUMEN

CLASIFICACIÓN DINÁMICA DE ESCENARIOS FUNCIONALES PARA EL DIAGNÓSTICO DE PROCESOS INDUSTRIALES

por

NATHALIE ANDREA BARBOSA ROA

Doctor en Ingeniería, Programa de Doctorado en Ingeniería – Ingeniería Eléctrica

UNIVERSIDAD NACIONAL DE COLOMBIA

Directores: Victor Hugo Grisales Palacio, Ph.D., Louise Travé - Massuyès, Ph. D.

Los avances tecnológicos de las últimas décadas han generado infraestructuras de producción y servicios que deben adaptarse rápidamente a un ambiente en constante cambio. Estos avances han cambiado la forma en la que las empresas obtienen la información acerca del estado de sus sistemas. Grandes cantidades de datos provenientes de diversas fuentes son recolectados y en muchos casos almacenados, lo que los convierte en un importante recurso para extraer información y conocimiento del proceso. En general, varios campos de aplicación han declarado la necesidad de herramientas que les permitan tratar con los datos que reciben de su proceso en forma eficaz y eficiente.

Las técnicas de minería de datos son ampliamente utilizadas en la industria actualmente sobre todo en lo que ha sido llamado *Industria 4.0*. La industria 4.0 o ciber-industria del futuro, se basa en el uso de dispositivos conectados (máquinas, sensores, actuadores) capaces de adaptarse rápidamente a las necesidades de producción y capaces de comunicarse en *tiempo real* con los entes supervisores del proceso. Los sistemas de supervisión ligados a esta industria no son los mismos que se tenían hace algunos años. Estos sistemas deben ser capaces de monitorear datos provenientes del proceso directamente y extraer de ellos la información sobre el estado del sistema. Estos datos de grandes dimensiones, suelen presentarse como un flujo continuo de información limitando la capacidad de procesar el mismo dato dos veces. Dado que actualmente el sistema en sí cambia a causa de las necesidades de producción y otras variables ligadas a su entorno, los sistemas de supervisión deben también ser capaces no solo de reconocer el estado actual del proceso sino aprender continuamente de sus cambios con el fin de poder monitorearlo de forma correcta y de mejorar el diagnóstico del estado actual, incrementando así la seguridad de la planta, minimizando los tiempos muertos y reduciendo costos de producción.

Los algoritmos de aprendizaje de máquina y reconocimiento de patrones han surgido como una solución para extraer de forma eficiente el conocimiento del proceso basándose en grandes cantidades de datos del proceso que pueden estar disponibles. El diagnóstico por reconocimiento de patrones es utilizado con frecuencia en sistemas industriales de control de calidad y monitoreo, sin embargo muchas de las técnicas actuales de diagnóstico basadas en datos consideran las variables solo como imágenes del estado del proceso en un momento dado ignorando muchas veces su evolución temporal, es decir, su dinámica. En dichos enfoques los conceptos que describen el estado del sistema (normal, en falla, avería, disfunción) son considerados como invariantes en el tiempo; es decir, la variación de las variables que describen el sistema indican directamente un cambio de estado, por ejemplo de “normal” a “en falla”.

Sin embargo, si dichos conceptos son considerados como variantes en el tiempo, no solo la información sobre el estado actual del sistema sino también aquella sobre cómo ha llegado allí son necesarias.

En esta tesis el trabajo se centró en desarrollar un algoritmo de agrupamiento dinámico de datos que no se limita a conceptos estáticos ni a conjuntos convexos y que a demás puede manejar distribuciones que evolucionan en el curso del tiempo. Este algoritmo puede ser utilizado en sistemas de supervisión de procesos pero su aplicación no está limitada a los mismos. En realidad, para poder incluir las relaciones temporales entre los datos, un método de extracción de características basado en la aproximación polinomial sobre ventanas de talla variable fue realizado. Una vez generadas las características estas sirven de entrada al algoritmo de agrupamiento que va a encontrar relaciones entre ellas basándose no solo en su similitud (distancia entre las muestras) sino tambien en la densidad de los grupos encontrados. Con el fin de facilitar el análisis y comprensión de los resultados del agrupamiento, un sistema a eventos discretos fue incluido en la salida del agrupador. Este sistema permite la visualización rápida del estado actual del sistema y la previsión de los posibles estados futuros incluyendo las restricciones temporales entre ellos.

Los aportes de esta tesis pueden entonces listarse en tres grupos:

1. Contribuciones al agrupamiento dinámico de datos: Los algoritmos de agrupamiento actuales basados en medidas de distancia asumen una representación convexa y por lo tanto son incapaces de hacer frente a las agrupaciones de datos no lineales o de formas alargadas. Por otro lado, los algoritmos basados en medidas de densidad capaces de implementar aprendizaje progresivo no consideran la posibilidad de detectar grupos que exhiben diferentes niveles de densidad. En la segunda parte de esta tesis se presenta un algoritmo que utiliza análisis basados en medidas de distancia y de densidad para agrupar muestras en clases de las cuales ninguna suposición ha sido tomada con respecto a la linealiadad o a la convexidad. Estas clases, que pueden presentar densidades variadas, pueden también presentar solapamiento entre ellas. El algoritmo desarrollado funciona en línea fusionando las etapas de aprendizaje y de clasificación, lo que le permite detectar y caracterizar de forma continua nuevos comportamientos y al mismo tiempo reconocer el estado actual del sistema. El algoritmo evoluciona de forma automática cuando estos comportamientos son detectados. Esta evolución se refleja en cambios en la estructura y/o parámetros del algoritmo. Los cambios en los parámetros van a generar el deslizamiento de una o varias clases y se dan en respuesta a una ligera variación en el concepto que se está siguiendo. Por ejemplo, en el ambiente industrial las normas pueden cambiar imponiendo estándares cada vez más altos para denominar un producto como de “alta calidad”. Los cambios en la estructura del algoritmo se generan por una de cuatro razones: la primera creación de clases. Si un comportamiento que no habia sido visto se produce, el algoritmo debe poder reconocerlo y caracterizarlo en una nueva clase. La segunda es la fusión de dos clases diferentes. Este fenómeno puede presentarse cuando dos comportamientos que se creían no relacionados evolucionan evidenciando una relación. La fusión de clases permite a un algoritmo ganar robustez con respecto al orden en el que las muestras son presentadas al sistema. La tercera razón es la separación de clases y se presenta cuando debido a la inclusión de una nueva variable o a la mejora de un sensor, nueva información es introducida en el sistema permitiendo distinguir dos comportamientos que antes parecían indistinguibles. La cuarta razón corresponde a la eliminación de grupos y se presenta cuando un antiguo comportamiento deja de ser significativo o representativo del sistema.

En resumen el algoritmo desarrollado, llamado ***DyClee***, tiene como características:

- es capaz de realizar el seguimiento del estado actual del proceso,
- es capaz de analizar grandes cantidades de datos en línea ya sea en forma de lote o en flujo continuo, de acuerdo con su naturaleza,
- es robusto en presencia de ruido,
- es capaz de detectar nuevos eventos,
- se adapta en forma automática para seguir la evolución del sistema,
- es capaz de manejar las limitaciones de recursos en memoria y en procesamiento,
- es capaz de detectar y caracterizar nuevos comportamientos del sistema sin afectar los comportamientos ya aprendidos,
- es capaz de reconfigurarse de manera automática; es decir, no es necesaria una etapa de reentrenamiento para cambiar la estructura del algoritmo.

2. Contribuciones a la extracción de características: los algoritmos de agrupación dinámica de datos que se encuentran en la literatura manejan en su mayoría características estáticas solamente, perdiendo la información relacionada con la dinámica de los cambios en el proceso. Los enfoques cualitativos para la extracción de tendencias son los únicos que tienen en cuenta la información sobre la evolución de las variables; sin embargo, al utilizar solamente representaciones cualitativas, la información cuantitativa sobre la magnitud del fenómeno descrito se pierde. En la tercera parte de esta tesis se presenta un nuevo enfoque para la extracción de características dinámicas. Este enfoque, basado en aproximación polinomial por trozos, permite representar comportamientos dinámicos sin perder la información relacionada con la magnitud y reduce a la vez la sensibilidad del algoritmo al ruido en la señal analizada. Para poder utilizar la aproximación polinomial en línea fue empleado un método de segmentación de señales en ventanas que se deslizan con la llegada de nuevos datos. En cada nueva ventana se realiza una búsqueda de puntos de inflexión para mejorar la calidad de la aproximación realizada. La validación de las aproximaciones encontradas se realiza comparando el error inducido por las mismas contra la varianza del ruido en la señal caracterizada, si este error es menor a la varianza la aproximación es validada y puede ser utilizada para describir el comportamiento actual de la señal.

3. Contribución en el modelamiento de sistemas a eventos discretos para sistemas evolutivos: Los resultados del algoritmo de agrupamiento son utilizados como base para desarrollar un modelo de eventos discretos del sistema. Este modelo adaptativo permite una abstracción de alto nivel del sistema en forma de un autómata temporisado cuyos estados representan los estados del proceso (fruto del reconocimiento efectuado por el clasificador) y sus transiciones, expresan la accesibilidad de los estados. Estas transiciones son identificadas con una posibilidad de transición (basado en las transiciones ya evidenciadas) y un tiempo estimado de transición presentado en forma de intervalo. Este modelo se construye automáticamente a medida que nuevos datos del sistema se reúnen y que de manera correspondiente el algoritmo de agrupación cambia su estructura.

RESUMÉ

CLASSIFICATION DYNAMIQUE POUR LE DIAGNOSTIC DE PROCÉDÉS EN CONTEXTE ÉVOLUTIF

par

NATHALIE ANDREA BARBOSA ROA
Doctorat de l'Université de Toulouse

Université Toulouse III - Paul Sabatier

Directeurs: Louise Travé - Massuyès, Ph. D., Victor Hugo Grisales Palacio, Ph.D.

Les avancées technologiques de ces dernières décennies ont généré des infrastructures et des services de production qui doivent s'adapter rapidement à un environnement changeant. Ces avancées ont changé la façon dont les entreprises obtiennent les informations sur l'état de leurs systèmes. Actuellement, de grandes quantités de données provenant de diverses sources sont collectées et stockées. Ces données sont une ressource importante pour extraire des informations sur les processus. L'exploitation des données et l'extraction de connaisances sont donc devenu un besoin dans de nombreux domaines d'application.

Actuellement, les techniques d'exploration de données sont largement utilisées dans l'industrie en particulier dans ce qui est appelé *Industry 4.0*. L'industrie 4.0 ou cyber-industrie de l'avenir, est basée sur l'utilisation d'objets connectés (machines, capteurs, actionneurs) capables de s'adapter rapidement aux besoins de la production et de communiquer en *temps réel* avec les organes de contrôle du processus. Les systèmes de contrôle liés à ce type d'industrie ne sont pas les mêmes qu'il y a quelques années. Ces systèmes doivent être capables de surveiller les données du processus directement et d'extraire des informations sur l'état du système. Ces données sont présentées généralement sous la forme d'un flux continu de mesures, limitant de ce fait la possibilité de traiter deux fois les mêmes données. Étant donné que le processus lui-même évolue en fonction des besoins de production et d'autres variables liées à son environnement, les systèmes de surveillance doivent également être en mesure de reconnaître l'état actuel du processus, mais aussi d'apprendre de ces changements en continu afin d'être en mesure de le surveiller et le diagnostiquer correctement.

Les algorithmes d'apprentissage automatique et de reconnaissance des formes sont apparus comme la solution pour extraire efficacement la connaissance des processus dans le cas où de grandes quantités de données sont disponibles et peu de connaissances formelles existent pour construire des modèles. Les méthodes de diagnostic basées sur la reconnaissance des formes sont souvent utilisées dans les systèmes industriels pour le contrôle de qualité et la surveillance. Cependant, de nombreuses techniques actuelles de diagnostic basées sur des données considèrent les variables comme des images de l'état du processus à un instant donné en ignorant souvent son évolution temporelle, à savoir sa dynamique. Dans de telles approches les concepts qui décrivent l'état du système (nominal, dégradé, défaillant) sont considérés comme invariants dans le temps. Les variations dans les données qui décrivent le système représent alors un changement d'état, par exemple de "nominal" à "défaillant". Néanmoins, si l'on considère que ces concepts peuvent varier dans le temps, les algorithmes doivent être revus.

L'objectif principal de cette thèse est de développer un algorithme dynamique de partitionnement de données (classification non supervisée ou "clustering" en anglais) qui ne se limite pas à des concepts statiques et qui peut gérer des distributions qui évoluent au fil du temps. Cet algorithme peut être utilisé dans les systèmes de surveillance du processus, mais son application ne se limite pas à ceux-ci.

Les contributions de cette thèse peuvent être présentées en trois groupes.

1. Contributions au partitionnement dynamique de données : les algorithmes de clustering actuels peuvent être classifiés selon la mesure de similitude qu'ils utilisent. Les algorithmes basés sur des mesures de distance supposent surtout que les données ont une représentation convexe et sont donc incapables de faire face à des groupes de données non-convexes ou des formes allongées. D'un autre côté, les algorithmes basés sur des mesures de densité capables de mettre en œuvre un apprentissage progressif n'envisagent que rarement la possibilité de détecter des groupes présentant différents niveaux de densité, ce qui pose un problème en diagnostic puisque les états fautifs sont en nombre inférieur. Dans la deuxième partie de cette thèse, un algorithme de partitionnement dynamique basé à la fois sur la distance et la densité des échantillons est présenté. Cet algorithme ne fait aucune hypothèse sur la linéarité ni la convexité des groupes qu'il analyse. Ces clusters, qui peuvent avoir des densités différentes, peuvent également se chevaucher. L'algorithme développé fonctionne en ligne et fusionne les étapes d'apprentissage et de reconnaissance, ce qui permet de détecter et de caractériser de nouveaux comportements en continu tout en reconnaissant l'état courant du système. Le classificateur évolue automatiquement lorsque ces comportements sont détectés, en changeant sa structure et/ou ses paramètres. La dérive d'un ou de plusieurs groupes s'effectue en réponse à une variation du concept qui est suivi (dérive conceptuelle). Par exemple, dans le milieu industriel, les normes peuvent changer, en imposant de nouveaux standards de qualité et de production de plus en plus élevés, pour pouvoir considérer un produit comme étant de "bonne qualité". Les changements dans la structure du classificateur peuvent apparaître dans quatre situations. La première situation est la création d'une classe. Si un nouveaux comportement se produit, l'algorithme doit être capable de le reconnaître et de le caractériser dans une nouvelle classe. La seconde est la fusion de deux classes. Ce phénomène peut se produire lorsque deux comportements que l'on croit sans rapport montrent une relation en évoluant. La fusion des classes permet à l'algorithme de gagner en robustesse par rapport à l'ordre dans lequel les échantillons lui sont présentés. La troisième est la séparation d'une classe et se produit lorsque, en raison de l'inclusion d'une nouvelle variable ou dû à l'amélioration dans un élément du processus (capteur par exemple), de nouvelles informations, permettant de distinguer deux comportements jusque là impossibles à distinguer, sont entrées dans le système. La quatrième situation correspond à l'élimination d'un classe et se produit quand un comportement n'est plus représentatif du système.

   L'algorithme développé, appelé ***DyClee*** (**Dy**namic **Cl**ustering algorithm for tracking **E**volving **E**nvironments), a plusieurs caractéristiques:

   - il est en mesure de suivre l'état courant du système,
   - il est capable d'analyser de grandes quantités de données en ligne, soit en batch ou en flux continu en fonction de sa nature,
   - il est robuste au bruit dans les signaux d'entrée,

- il est capable de détecter et de caractériser les nouveautés sans affecter les comportements déjà appris.
- il adapte automatiquement ses parametres afin de suivre l'évolution du système,
- il gére des contraintes sur les ressources mémoire et les limitations liées au processeur,
- il se reconfigure automatiquement, c'est-à-dire qu'il n'a pas besoin d'une nouvelle étape d'entraînement pour modifier la structure du classificateur.

2. Contributions à l'extraction de caractéristiques : la plupart des algorithmes de partitionnement dynamique de données de la littérature gèrent seulement des caractéristiques statiques, perdant ainsi l'information relative aux changements dynamiques dans le processus. Les approches qualitatives qui permettent d'extraire les tendances d'un signal prennent en compte l'information sur l'évolution des variables. Cependant, en utilisant uniquement des représentations qualitatives, des informations quantitatives sur la magnitude du phénomène sont perdues. Dans la troisième partie de cette thèse une nouvelle approche permettant d'extraire des caractéristiques dynamiques est présentée. Cette approche, basée sur une approximation polynomiale par morceaux, permet de représenter des comportements dynamiques sans perdre les informations relatives à la magnitude et en réduisant simultanément la sensibilité de l'algorithme au bruit dans les signaux analysés. Pour pouvoir se servir de l'approximation polynomiale en ligne, une méthode de segmentation à fenêtres glissantes des signaux a été implémentée. Dans chaque nouvelle fenêtre, une recherche de points d'inflexion est effectuée afin d'améliorer la qualité de l'approximation. La qualité de l'approximation polynomiale est validée en comparant l'erreur induite par l'approximation à la variance du bruit dans le signal caractérisé, si cette erreur est inférieure à la variance, l'approche est validée et peut être utilisée pour décrire le comportement sur le segment de signal considéré.

3. Contributions à la modélisation de systèmes à événements discrets évolutifs : les résultats de l'algorithme de partitionnement sont utilisés comme base pour l'élaboration d'un modèle à événements discrets du processus. Ce modèle adaptatif offre une representation du comportement du processus de haut niveau sous la forme d'un automate dont les états représentent les états du processus appris par le partitionnement jusqu'à l'instant courant et les transitions expriment l'atteignabilité des états. En fait, la représentation graphique du modèle présente les états passés et l'état actuel du système ainsi que les transitions possibles entre ceux-ci. Ces transitions sont identifiés avec une possibilité de transition (calcul basées sur les transitions déjà produites) et l'estimation du temps requis pour la transition (sous forme d'intervalle). Ce modèle est construit automatiquement et se met à jour lorsque de nouvelles données sont collectées en fonction des résultats du partitionnement.

A mi familia

'One often meets his destiny on
the road he takes to avoid it'

"On rencontre sa destinée souvent par
des chemins qu'on prend pour l'éviter"

Jean de La Fontaine

RECONOCIMIENTOS

Quisiera agradecer en primer lugar a Dios por permitirme tomar este camino que tanto me ha enseñado sobre mi misma. A mis padres, por su amor incondicional, por su paciencia y su gran ayuda en la recta final; soy consciente que no hubiera podido terminar sin ustedes. A mi familia, por sus infinitas oraciones y por hacerme sentir cerca incluso cuando miles de kilómetros nos separan. A mis mejores amigos: Burris, Bombi, Chu, Yu y Adri por ser mi bastión, por ayudarme siempre a encontrar una sonrisa, en fin, por ser los mejores amigos que nunca pedí pero que me regocijo de disfrutar. A Julien por llegar a mí cuando mi avión se estrelló en el desierto. *Julien* gracias por compartir conmigo tus amigos, tu familia; pero sobre todo gracias por apoyarme en esos momentos críticos al final de mi doctorado.

Quisiera agradecer al ingeniero Hernando Díaz por iniciarme en la investigación, forjarme metodológicamente y mostrarme que se puede vivir aprendiendo toda la vida. Al ingeniero Victor Hugo Grisales, por aceptarme cuando recurrí a él pidiendo ayuda y brindarme la que sería la mejor solución. *Ingeniero* gracias por mostrarme que se ganan mas favores con rosas que con espinas. A Louise Travé-Massuyès por darme la oportunidad de conocer lo que es la investigación en Francia, por su acogida en Toulouse, por su motivación, guía y apoyo. *Louise* gracias por creer en mi autonomía y mi capacidad. A Quentin Gaudel y Thomas Monrousseau por compartir conmigo mis aventuras y desventuras a nivel científico y personal y toda la dicha y desdicha de ser doctorantes DISCO. *Dr. T* y *Dr. Q* gracias por nuestras largas conversaciones filosóficas sobre ciencia, tecnología, sobre el bien y el mal, pero sobre todo gracias por estar ahí. A Q gracias también por compartir la oficina conmigo, por ayudarme en mi tesis y por involucrarme un poco en la tuya.

A DISCO, especialmente a Marie-Veronique, Audine, Euriell, Pauline, Elodie y Yannick, por su acogida, su apoyo y por intentar salvar el mundo; un proyecto a la vez. Quisiera también agradecer a los pasantes que hicieron de nuestras tardes de Dumbash más amenas, que trajeron alegría, vida y entusiasmo al equipo: a Frédéric, Antonin, Marwa y un poco a Pascal también.

Finalmente quisiera agradecer a mis jurados: Claudia Isaza, Rafael Gouriveau y German Zapata por la cuidadosa lectura de este manuscrito, por sus comentarios y preguntas que me permitieron mejorar este documento. A Marie-Veronique por haberme hecho el honor de ser presidente de jurado, por sus comentarios y apoyo a lo largo de esta tesis.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

The next list describes several symbols that will be later used within the body of the thesis.

Abbreviations

A-list Active μ-clusters list

O-list Less-active μ-clusters list

DyClee **Dy**namic **C**lustering algorithm for tracking **E**volving **E**nvironments

μ-cluster Micro-cluster

DES Discrete event system

NN Nearest neighbours

Mathematical Nomenclature

Δt Time differential

ΔX Data set differential

ι Subset of clocks to be reset

$\mathcal{A}$ Automaton

$\mathcal{A}_t$ Timed Automaton

$\mathcal{CK}$ Set of clocks

$\mathcal{D}$ Set of the cluster's median time of stay

$\mathcal{S}$ Set of States

$\mathcal{S}_0$ Set of initial states

$\mathcal{S}_m$ Set of marked states

μC_z zth Micro-cluster

ϕ Fraction of feature range used as μC size

$\Upsilon(\mathcal{CK})$ Set of timed constraints

φ	Number of allowed unconnected dimensions
ξ	low-density Threshold
a	Slope of a linear function
C_z	Cluster center
c_z	Center of μC_z
c_z^j	Coordinates of the μC_z center in its jth dimension
Cl_i	Cluster i
CT_z	Characteristic tuple representing μC_z
d	data sample dimension
D_z	Density of μC_z
Dis_{iz}	Distance between a data sample x_i and the cluster center C_z
E	Set of events
$f\left(t, t_{lk}\right)$	Forgetting function depending on t and t_{lk}
h_i	Measured level of the tank i
$id_{\mu C_z}$	Unique identifier of μC_z
k	Number of classes or clusters
L	Set of discrete values used as labels for the clusters
L_1	Manhattan norm
L_2	Euclidean norm
L_p	Minkowsky norm
L_∞	Infinity norm or maximum norm
lab_z	Label assigned to μC_z
LS	Vector of features linear sum
m	number of samples in a dynamic feature
N	Number of samples in the data set
n	Sample identification number related to the order of arrival
n_z	number of data samples mapped into μC_z
Q_{out}	Tank outflow
s_0	Initial state

S_z^j	Size of the μC_z in its jth dimension
SS	Vector of features sum of squares
T	Time interval
t	Time instant measured in seconds unless otherwise specified
$t_{\sim f}$	Forgetting function start time, before this time $f(t, t_{lk}) = 1$
t_{global}	Time between two runs of the density-based stage
t_{lz}	time of last update of μC_z
t_{sz}	μC_z time of creation
$t_{w=0}$	Time when the forgetting function reaches zero for the first time
Tf	Transition function
ts_i	Time of arrival of the data sample x_i (time stamp)
V_z	Hypervolume of μC_z
X	Data set
$x(t)$	Dynamic data sample
x_i	ith data sample
x_i^j	jth Static feature of the data sample x_i
$x_i^j(t)$	jth Dynamic feature of the dynamic data sample $x_i(t)$
$\mathbb{F}$	Domain of features

Chapter 1

Introduction

1.1 Scientific context

Technological advances of past decades have resulted in production and service infrastructures highly adaptive to a constantly changing environment. These advances had changed the way enterprises get information about the state of their systems [Bar *et al.* 15]. Huge amounts of data, arising from various sources, are generally collected and they are available for further analysis. As reported by the International Data Corporation (IDC), the volume of data generated in the world is doubling in size every two years. In 2013 this volume were estimated around the 4.4 trillion gigabytes and is expected to grow up to 44 trillion by 2020 [Tur *et al.* 14].

In general, several applications and fields urgently need efficient and effective tools and analysis methods for dealing with the ever-growing amount of data [Gam *et al.* 14a]. Moreover, the use of data mining and data analysis techniques was recognised as necessary to increase business opportunities, to improve service and to maintain competitiveness in today's business world [Ang 01]. In the field of supervision, the increasing amount of process data collected and stored has led to an equally increasing interest in the research of supervision techniques that:

- process large dimensional data, sometimes coming in the form of streams.
- learn from the data and constantly adapt the targeted model to the evolution of the system (present as change in the data distribution).
- cope with infrequent behaviors represented by sparse data.
- find meaningful correlations, patterns and trends from the data sets.
- cope with unlabeled data.

This interest is shared in industrial and academic scenarios since fast and accurate diagnosis of the system state can increase the plant operation safety, minimize the shut-off time and reduce costs. The need of techniques capables to adapt in the presence of unknown behaviors is also crucial for many applications in which faulty data is rare or even dangerous to gather [Gou *et al.* 13]. Among various methods, approaches that make direct use of data have been described to be most useful for industrial applications [Mau *et al.* 10]. Machine learning algorithms and pattern recognition techniques emerge as a feasible solution to extract efficiently knowledge from large amounts of available process data. Pattern recognition based diagnosis is frequently used in industrial systems for quality control (in normal operation) and monitoring. In the case of detection of dangerous states, the monitoring function automatically initiates an appropriate reaction.

Most of the data-based fault diagnosis methods consider data as a set of instant objects without taking into account their temporal evolution, i. e. their dynamics. In those traditional approaches, systems are considered as time invariant, so changes in the feature pattern (coming from physical or analytical sensors) denote a state change. Nevertheless, if the system is no longer considered as time invariant, not only the information of current values but also how those values change in time are needed in order to establish the current state. There exist several applications in a variety of domains for which the order of occurrence of data samples determines the state of the system, such as: network intrusion detection, customer services (relationship,segmentation, retention, complains management), fraud detection, conditional maintenance (e.g. tool wear monitoring), etc.

This thesis presents a dynamic clustering approach suitable for monitoring the state of time varying systems. This approach is:

- able to track the current state of the process,
- suitable to large amounts of data; the data sets are analyzed online and, according to their nature, in batch or online mode; if the data set is small enough the analysis of the hole set can be done at once (offline),
- robust to the presence of noisy data,
- able to detect novel events,
- adapted to follow system evolution online; the generated clusters change when data distribution changes without retraining,
- able to manage storage and computational limitations,
- able to learn new system behavior online, without affecting the previously learned behavior,

- self-re-configurable, clusterer structure changes automatically in presence of system changes.

1.2 Motivation

This thesis is the result of my personal desire to contribute to the development of my country. The economic and social impact of this work can be established through the analysis of the implications that the application of the results of the thesis could have. On the economic side, in Colombia, 36% of GDP (Gross Domestic Product) comes from the industry. Given this number, any research focused on the industry can contribute in order to boost the economy in Colombia is promising. One big difference between the so-called advanced economies and the developing economies is the investment on research and the frequency of academy-industry collaboration projects. On a social perspective, businessmen and citizens of Bogotá believe that, in order to improve education, it is necessary to join higher education programs with productive sectors [Cam 11]. The knowledge gathered through this thesis could improve the University master program in Industrial Automation, and could lead to an improvement in education. Moreover, the developed algorithm could lead to an improvement in supervisory systems, a field in which Colombia has little research background. This improvement would imply less faults and would reduce human injuries and material losses. This motivation fits with the regional vision of Bogotá.

Bogotá is the capital of Colombia and the focus of research and innovation in the country. Generating 31% of national GDP,the region comprising Bogotá and Cundinamarca (Bogotá-Region) is the driving force of the Colombian economy thanks to it size, the dynamics of its productive activities, job creation and strength of its business. Being the largest market in Colombia, its principal strength is to have a diversified production structure, in which services are the predominant activity (56.2%), followed by trade (14.5%), industry (11.3%) and construction (4.6%) [Cam 15]. Bogotá-Region is the most enterprising region in Colombia, with more than 384 thousand businesses in total, and an average of 76 thousand new businesses per year.

The capital aims to structure the policy 'Bogota 2025', a policy of 'Intelligent Strategic Specialization' in Bogotá-Region through a productive vocation based on the keystones of knowledge, science, technology, research and innovation. The so-called *key enabling technologies*, capital and knowledge-intensive technologies associated with a high degree of research and development and innovation, are the advocacy of an 'intelligent development' in the region. Colombian specialists in business and politics have a clear vision of Bogotá-Region in 2025 as a leader of development based on knowledge, research and technological advancement [Cam and Alc 15]. Among the possible scenarios of what could happen in Bogotá in the next

10 years, innovation can come from the public sector, in form of strategic alliances between the state, the businessmen and the academia, or from the private sector, in form of a research platform between the private sector and the universities [Cam and Alc 15].

1.3 International scientific collaboration

This work was developed in collaboration between the Automation research group at Universidad Nacional de Colombia (GAUNAL) and the research group in Diagnosis and Supervisory Control (DISCO) from the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS).

GAUNAL is an interfaculty research group with valuable experience in the field of diagnostics applied to electric power generation and transmission systems. Industrial applications of its research can be found in the area near the city of Medellin. At present time, the group vision includes to reach similar achievements in the metropolitan area of Bogotá and to build the foundations of a future graduate course devoted to monitoring and diagnosis.

The DISCO research team conducts a rich and varied methodological research in the field of automated diagnosis and prognosis. Relying on formalisms from both the field of Automatic Control Artificial Intelligence, DISCO researchers develop expertise at the intersection of these two fields. Model-based methods as well as machine learning and data mining methods are investigated to provide original diagnosis and prognosis solutions. The experience in considering varied systems and the multidisciplinary approach are the trademarks of the team, whose results are recognized in both fields.

1.4 Structure of the Thesis

This thesis is organized in three main parts. The first part present as preliminary information the thesis context, the addressed problem and the state of the art in classification. The second part presents the proposed dynamic clustering algorithm capable of clustering *static and dynamic objects* and illustrates its properties using toy examples over static data. The third part present the use of the dynamic clustering for monitoring dynamic processes illustrated in an industrial benchmark. The content of each part is further explained below.

The Part I is organized as follows:

In chapter 2, a general framework on fault diagnosis techniques is given. Starting with the general structure of a diagnostic system and followed by some definitions, the role of fault diagnosis in supervision is explained. This is followed by a brief description of the desired features of a supervisory system and a classification of the techniques that have been proposed in order to solve the fault diagnosis problem.

Chapter 3 introduce the data classification problem as has been considered in classical approaches and then introduces the dynamic classification problem. It will be seen that dynamic classification may refer to the dynamism of the classified data or to the classifier structure itself. An state of the art on dynamic classification is presented from which, arguably, the need of a fully dynamic clustering algorithm can be outlined.

Part II is organized as follows:

Chapter 4 introduces the dynamic clustering algorithm developed throughout this thesis. This algorithm make use of the advantages of both distance- and density-based clustering algorithms to resolve classification problems that can not be solved by using just one of these methods. The distance-based stage if further detailed in Chapter 5 where an analysis over the distance measures bets suited to work with supervisory problems is also performed.

Chapter 6 introduces the density-based clustering stage. In this stage the output of the distance-based stage is analyzed in order to find clusters of any shape and size, including non-convex high overlapping sets. The part II ends with a variety of test performed over static data showing that the developed algorithm outperforms several well known clustering algorithms.

Part III is organized as follows:

Chapter 8 presents this thesis proposal to generate *dynamic features* capable of characterizing dynamic data. This dynamic features can be treated in the same way that static features by the developed algorithm allowing clustering of dynamic objects. Chapter 9 then introduces how the clustering results can be used to learn a time dependent discrete event model of the process under analysis.

The final chapter of this part, Chapter 10, shows the application of the developed algorithm on two industrial benchmarks. The first one is a pilot process describing a thermal power plant on a reduced scale found at the Lille 1 University - Science and Technology. The second one, a pilot plant of a continuous stirred tank heater located in the Department of Chemical and Materials Engineering at University of Alberta in Canada.

Finally, the conclusions of this thesis and recommendations for future work are presented in Chapter 10.3.

1.5 Contributions

The main contributions of this thesis can be summarized in terms of contributions on dynamic clustering of static and dynamic objects, on dynamic feature extraction and on discrete event modeling for evolving systems:

- *Contributions on dynamic clustering:* Current distance-based clustering algorithms rely

on convex representation and are therefore unable to deal with non-linear elongated clusters. On the other hand, density-based algorithms that currently handle incremental learning do not handle cluster distributions with varied densities. In the part II, an algorithm that uses distance- and density-based analyses to cluster non-linear, non-convex, overlapped data distributions with varied densities is proposed. This algorithm, that works in an online fashion, fusions the learning and classification stages allowing to continuously detect and characterize new behaviors and at the same time recognize the system current state.

- *Contributions on feature extraction:* Dynamic clustering algorithms found in the literature mostly handle static features only, losing the information related to the dynamics of the process changes. Qualitative approaches to trend extraction are the only ones to consider and keep information about the evolution of a variable, however, they serially lose the quantitative information about the magnitude of the described phenomenon. In Chapter 8, a novel approach to dynamic feature extraction is presented. This approach based on polynomial approximation allows to represent dynamic behaviors without losing the magnitude related information and to reduce the algorithm sensitivity to noise at the same time.

- *Contributions on discrete event modeling for evolving systems:* In Chapter 9 the found clustering results are used to develop a discrete event model of the system. This adaptive model give a high level abstraction of the system with information about the system past and current states along with the possible transitions identified by the time of transition and the time constraints between two states.This model is built automatically as new data of the system is gather.

As a result of these contributions the following publications were generated:

- N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *Trend-Based Dynamic Classification for on-line Diagnosis of Time-Varying Dynamic Systems.* In SAFEPROCESS 2015, Proceedings of the 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, pages 1224–1231. IFAC, 2015

- N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *DyClee: Dynamic clustering for tracking evolving environments.* Pattern Recognition, 2016. Submitted

- N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *Dynamic Clustering as a Tool for Monitoring Evolving Systems.* In DX-2016, Proceedings of the 27th International Workshop on Principles of Diagnosis. DX, 2016

- N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *Dynamic clustering for process supervision.* In XVII CLCA, Latin American Conference of Automatic Control. IFAC, 2016

- N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *A novel algorithm for dynamic clustering: properties and performance.* In ICMLA 2016, 15th IEEE International Conference on Machine Learning and Applications. IEEE, 2016

The developed algorithm was implemented in Python and is open for academic purposes. An introduction to the algorithm properties and use is given in Appendix B.

Part I

Preliminaries

This part aims at presenting the scientific context over which this thesis was developed. To do so, first a framework of fault diagnosis is presented in Chapter 2. This framework starts with the field definitions in order to establish a common vocabulary and then focuses on the desired characteristics that a supervision system should have. It culminates with the classification of fault diagnosis techniques. Then, Chapter 3 introduces the key concepts covered during this thesis. These concepts, mainly related to the classification of static and dynamic objects, are introduced using practical examples. Then the chapter focuses in dynamic classification, the desired evolution of the dynamic classifier structure and finalizes with an state of the art in dynamic classification techniques.

Chapter 2

Fault diagnosis framework

The increasing complexity of automated systems has been accompanied by the ever increasing demand of availability and safety in industrial plants [Tos 11]. The implementation of a fault detection and diagnosis system is one way to achieve these goals. By continuously monitoring the evolution of the system, early detection of deviations of the process behavior can be discovered and functional scenarios could be establish. A functional scenario is the sequence of actions in a system necessary to perform a task. To characterize the functional scenarios every possible behavior of the system must be found and represented as a sequence of states, including their relation with the system inputs. The detection and further maintenance is useful to prevent a failure before it appears. In order to avoid system breakdowns, production deterioration and/or damage to machines or humans, faults have to be found as soon as possible and the decisions involving such events have to be made quickly. When a deviation is considered abnormal (fault detection), an indicator of fault is generated (symptom), then the symptoms must be appropriately interpreted to find the origin of the anomalies (fault diagnosis). Finally, a decision must be made (decision-making) and applied in order to bring back the system to its normal behavior, see Figure 2.1. A system that includes detection, isolation and identification or classification skills is referred to as an FDI (Fault detection and isolation) system.

This chapter explores fault diagnosis in supervision. Fault detection and diagnosis covers a wide variety of techniques ranging from the use of fault trees, analytical approaches, knowledge based systems and neural networks in more recent studies. Before going further and in order to overview the FDI framework, some definitions must be established as a solid cornerstone for the diagnosis of faults.

2.1 Fault diagnosis generalities and definitions

The overall concept of fault diagnosis consists of three essential tasks performed over the system: fault detection (deciding whether or not a fault has occurred), fault isolation (deciding on which components the fault has occurred. Localization and classification) and fault analysis or identification (assessment of type, magnitude and causes) [TM 14]. To detect a failure behavior, normal behavior should be defined first.

Definition D2.1. Normal operation of a system: A system is said to be in normal operating conditions when the variables that define the process (state variables, output variables, input variables, system parameters) remain close to their reference/nominal values. The system is said to be faulty otherwise [Tos 11].

Considering the above, we can define a fault and a failure as follows.

Definition D2.2. Fault: A fault is defined as an unauthorized deviation of at least one property of a variable from its acceptable behavior (nominal situation) [Ise 05]. Faults can occur in the sensors, actuators or in the process itself.

Therefore, a fault is defined as an abnormality in the process or symptom. If a faulty state lasts long enough, it may lead to a system malfunction or failure.

Definition D2.3. Failure: A failure implies the alteration or cessation of the ability of a device to perform the required function within the range given by the technical specifications. Failures are malfunctions of the system.

It is clear that failure implies fault. Nevertheless, a fault does not necessarily imply a failure because the device may very well continue to provide its primary function [Tos 11]. If the failure lasts long enough, it may lead to a system breakdown.

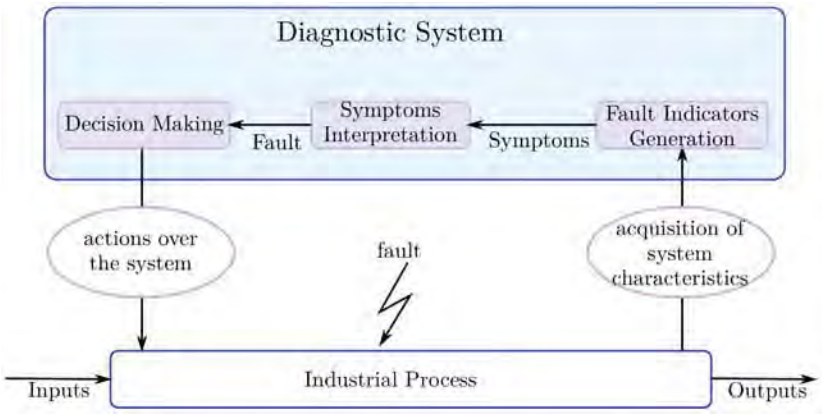


Figure 2.1: General structure of a diagnostic system. Based on [Tos 11]

Definition D2.4. Breakdown: A breakdown is the inability of a device to perform a required function.

It is clear that the diagnosis process should be able to detect and locate a fault before it could lead to a failure or system breakdown. Furthermore, not only the detection of the fault is important, but also distinguishing between the faults and their effects. Fault diagnosis has to trace back the cause-effect relationships from the measured variables when they prove to be different from the nominal values, in order to find the primary cause of the deviation [Bla 03]. Generally, there are three classes of elements that lead to failure behavior and that have to be addressed:

1. **Unknown inputs:** This phenomenon arises when disturbances enter the process through one or more parameters. In this case, an undesirable behavior due mostly to unmodeled dynamics, is observed. An example of this kind of faults is a change in the reactant concentration from its normal value in a rector feed it the concentration was not considered in the process model [Ven 94].

2. **Malfunctions in the process:** They refer to structural changes in the process itself. These changes occur due to hard failures in the equipment. An example of this kind of malfunction could be the disconnection of a system component.

3. **Malfunction in sensors or actuators:** Most faults occur due to sensor or actuator faults. An instrument fault could cause a deviation in plant state variables. If the fault is not detected on time, the variable deviation can exceed acceptable limits, thus causing an undesirable behavior. Examples of structural changes are the blocking of a valve or the drift in sensor measurements.

Faults may manifest in an abrupt form, an incipient form or an intermittent form. Fault behavior can also be classified for model purposes as additive or multiplicative [Ger 88]. Additive measurement faults are discrepancies between the measured and the true values of plant variables mostly due to sensor biases. Additive process faults are disturbances (unmeasured inputs) acting on the plant. Finally, multiplicative process faults are changes of the plant parameters, generally caused due to component deterioration.

2.2 Desired features of a Supervision system

Modern methods for supervision and fault diagnosis must have some desirable characteristics as stated in [Ven *et al.* 03, Rib and Bar 11]:

a. Quick detection and diagnosis: The system should detect small abrupt faults as soon as possible with a tolerable performance. There is a trade-off between robustness and time response.

b. Isolability: The system should have the ability to distinguish between different faults. There is also a trade-off between isolability and modeling uncertainty rejection.

c. Robustness: The system should be able to hold an acceptable performance against noise and uncertainties. Like faults, disturbances and model uncertainties change the plant behavior. Disturbances are usually represented by unknown input signals that have to be added up to the system. Model uncertainties change the model parameters in a similar way as multiplicative faults.

d. Novelty Identification: The system should have the ability of classifying the actual process state as normal or abnormal. If the process is functioning abnormally, the cause must be recognized as a known or unknown (novel) malfunction.

e. A priori error estimation: The system should provide an estimate on the classification error that can occur. These kind of information can increase the confidence of the user in the supervision system.

f. Adaptability: Since the process is prone to change, the diagnostic system should be adaptable to changes.

g. Cause-effect reasoning: It is desirable that the system not only detects a faulty condition, but also explains how the fault originated and propagated to the actual situation.

h. Modelling requirements: The modeling effort should be as minimal as possible, in order to achieve fast real-time diagnostic.

i. Storage and Computational requirements: The system should be memory and computational efficient. The diagnosis process present a trade-off between computational complexity and storage requirements, which is not easy to achieve.

j. Multiple fault identification: The system should be able to identify multiple faults occurring at the same time. The ability to identify multiple faults from different sources is hard to achieve. Since faults can show a synergistic behavior, a diagnostic system may not be able to use the individual fault patterns to model the combined effect of multiple faults.

k. Learning: The system should be able to learn and refine its knowledge and diagnostic capabilities as more data becomes available (online).

l. **Self-reconfiguration:** The system should also evolve preferably autonomously otherwise should have minimal reconfiguration capabilities.

2.3 Classification of Fault diagnosis Techniques

Several diagnosis techniques have been suggested in order to solve the fault diagnosis problem (see Figure 2.2). The sequel presents a rough classification of these techniques.

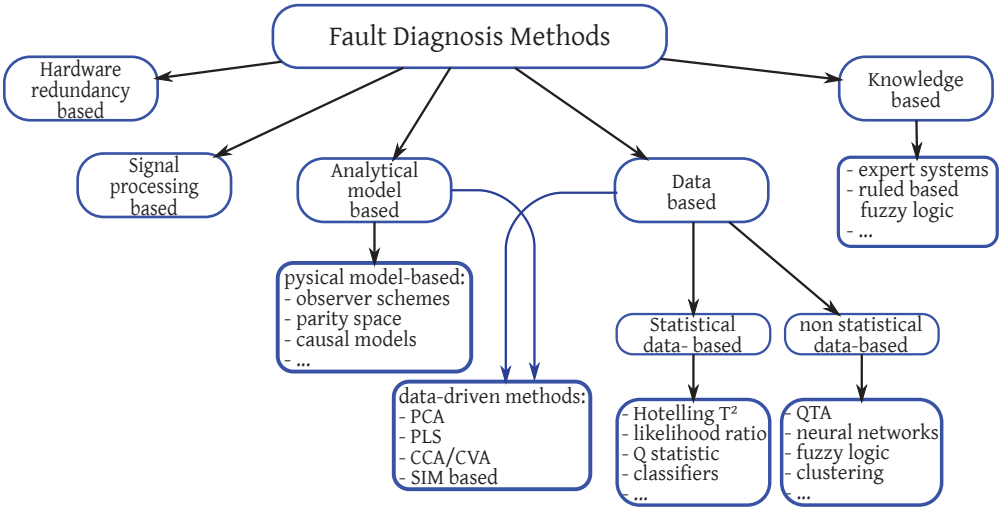


Figure 2.2: Classification of fault diagnosis methods

2.3.1 Hardware redundancy based fault diagnosis

Hardware redundancy accomplishes almost all of the previous characteristics. In hardware redundancy, process components are reconstructed using identical (redundant) hardware components. In this case, a fault is detected when a component and its redundancy have different outputs, therefore, component fault detection is straightforward. Fault isolation is generally conducted with a voting process which requires at least two redundant items. Besides fault detection, the other main advantage of this scheme is its high reliability. On the other hand, the use of this strategy results in high costs [Din 08]. Nowadays, this type of redundancy is only used in process key components or critical systems such as the space-shuttle [Zim *et al.* 11] (which uses four redundant computers for the primary flight system) or the Airbus A320 [Tra *et al.* 04] (which uses two types of computers, each one composed of two identical units).

2.3.2 Signal processing based fault diagnosis

Signal processing aims to extract fault information directly from process signals. Fault diagnosis can be achieved by a suitable signal processing on the assumption that some process signals carry information about the faults. This information takes the form of symptoms (see Figure 2.1). Common symptoms can be time domain functions or frequency domain functions. Some time domain symptoms are magnitude, limit value, trends, mean values (arithmetic or quadratic),etc. Examples in the frequency domain are spectral power densities, frequency spectral lines, cepstrum analysis, etc [Din 08, Din 14]. Similar to statistical data techniques, signal processing techniques are mainly used for processes in steady state and have limited efficiency for dynamic systems, which have a wide operating range.

2.3.3 Analytical model based fault diagnosis

The analytic redundancy strategy is the foundation for the Model-Based Techniques which are based on the availability of a priori knowledge about the process represented by some kind of model (quantitative or qualitative) in the form of a mathematical description of the process dynamics. A model-based fault diagnosis scheme consist of two parts: the *generation* of so-called residuals, which capture the difference between the measured process variables and their estimated values (found using the process model) and *residual evaluation* and decision making [Din 14]. Processes are exposed to disturbances and uncertainties making almost impossible to find their exact models. A central problem in the model-based fault diagnosis techniques is the extraction of the fault information from residual signals corrupted by model uncertainties and unknown disturbances [Din 08].

2.3.4 Data-based fault diagnosis

In contrast to model-based techniques, in data-based approaches (also called process history approaches), no a priori knowledge is required. Instead, a large amount of historical process data must be available. In industrial processes, data directly arise from process measures in form of voltages, currents, pressures, temperatures, quality measures, data vision feedback, etc.

The first problem consist in generating relevant features. Features can be found by using either statistical or non statistical methods. Accordingly, data-based fault diagnosis can be roughly classified into statistical data-based fault diagnosis and non statistical data-based fault diagnosis. Data-based schemes principally consist of two phases: the training stage, in which historical data sets are transformed into a diagnostic classifier, and the online running stage, in which the measurements data are processed through the classifier achieving a reliable

fault detection and identification [Din 14]. Statistical approaches are mainly used in static processes and are often limited in their efficiency for complex processes [Din 08].

2.3.5 Data-driven fault diagnosis

Data-driven methods assume (analytical) model-based methods in the statistical data-based framework [Din 14]. Different from analytical model based methods that require a priori knowledge of the first principles physics of the process, data-driven methods perform analysis over process historical data to find the parameters of the chosen model structure. Among the existent data-driven techniques principal component analysis and partial least squares regression are the most popular. In those techniques processes monitoring scheme for a large-scale system can be improved significantly by computing some meaningful statistics [Chi *et al.* 01]. The strength of data-driven techniques is their ability to reduce data dimension while keeping the important information.

2.3.6 Knowledge based fault diagnosis

Knowledge-based fault diagnosis is based on a qualitative model which represents a priori knowledge of the process under monitoring. Fault diagnosis is then achieved by running well developed search algorithms. The core of a knowledge-based fault diagnosis system is an expert system which consists of (i) a knowledge base (ii) a data base (iii) an inference engine and (iv) an explanation component. Knowledge-based fault diagnosis techniques are receiving increasing attention for dealing with fault diagnosis in complex technical processes [Din 14].

Recent developments of hybrid methods focus on achieving most of the desired characteristics shown in subsection 2.2 to exploit the strengths of two or more methods, so that their weaknesses are overcame.

Chapter 3

Data Classification for monitoring

To estimate the current operating condition of the system is the first step to predict its future state (prognosis). The assessment of the current functional scenario (monitoring) of the plant is basically a pattern recognition problem. The recognition involves learning similarities and differences of patterns that are abstractions of objects in a population of non-identical objects. Classification methods have been widely used for this purpose.

This work focuses in the unsupervised classification of data for monitoring and fault diagnosis. The previous chapter presented the fault diagnosis framework, this chapter aims to provide a general overview of data classification and its application to process monitoring. The classes or clusters represent functional or behavioral process states and their recognition, as said before, can be useful for monitoring and diagnosis.

Since a variety of notations can be found in the literature of both fields, data-based fault diagnosis and data classification, the main notations and definitions used in this thesis are first defined.

3.1 Preliminary definitions

This section give the definitions of the main concepts used throughout the thesis. As stated in the previous chapter, the observation space is usually of high dimension and should be transformed into a feature space when similarities, associations and relationships between objects make knowledge extraction easier.

The main focus of this chapter is the classification of objects' current state. Classification is the task of assigning objects to a group characterized by a given concept. A concept, as defined in the oxford dictionary, is *An idea or mental image which corresponds to some distinct entity or class of entities, or to its essential features.* Let us take as example a crop of apples.

Example 3.1. Apples are described by numerous characteristics as color, size, variety, etc. These characteristics represent any type of apple, for example, a specific apple can be described by the concept 'green' or more in the more elaborated 'green granny smith apple'. The set of all these characteristics describe the current state of the apple and ,since it is a *permanent state*[1], it describes the object itself (see Figure 3.1).

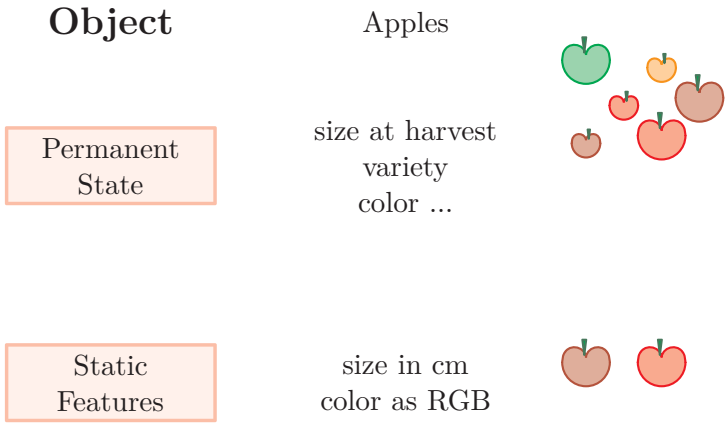


Figure 3.1: Data representation of a static object

If one is interested in selling red big apples, not all the apple's characteristics need to be considered. The selection of the characteristics containing the necessary information for classifying the apples is called feature extraction.

Definition D3.1 (Feature extraction)**.** The process of reducing the dimensionality of the object's characteristics in a way that retains the information discriminating different objects with respect to a given concept is called feature extraction.

Consequently a feature can then be defined as follows.

Definition D3.2 (Feature)**.** A feature is a distinctive characteristic of an object that sets it apart from similar objects with respect to a given concept. This characteristic does not need to be directly observable.

In the apple sell example the classification can be made using as features the size of the apple and its color represented in their red, green and blue components. These features are called static since they represent permanent characteristics of the object. Figure 3.1 illustrates this concept. The concept 'big red apple' denotes the set of all apples with size over $9cm$ and color redder than the RGB (red, green ,blue) color $[224, 50, 60]$. It is worth noting that one characteristic can be described by several features as is the case of the color

[1]This state can be called as *permanent state* since, once harvested, these characteristics do not change, i.e. a green apple cannot become red and vice versa.

which is described by three different features. The mathematical description is shown in equation (3.1).

$$big\ red\ apple\{x = [x^1, x^2, x^3, x^4] \ \mid x \in X \ x^1 \geq 9, x^2 \geq 224, x^3 \leq 50, x^4 \leq 60\} \tag{3.1}$$

where X is a data set (as defined later).

The concept of 'big red apple' can be used to illustrate another important definition addressed by this thesis, *the concept drift*.

Example 3.2. Let us think in the apples crop of two different periods. The first period had suffer an intense drought while the second does not. Intuitively, the apples from the drought period are smaller than those from the other period hence drifting the concept of 'big red apple'.

Definition D3.3 (Concept drift)**.** Given a data set X and a set of concepts L following a distribution $p(x, l)$, concept drift denotes the change in the current distribution $p(x, l)$ with respect to past distributions.

When people think about concepts two more actions can be directly associated to them, *concept creation* and *concept elimination*. A concept creation example is the bravo apple, introduced in 2014 in western Australia. This apple, result of the cross between two different apple varieties, has a striking burgundy color unlike any other apple currently available in Australia. Formal description of both concept creation and elimination are given below.

Definition D3.4 (Concept creation)**.** Given a data set X following a distribution $p(x)$ and a set of concepts L following a distribution $p(x, l)$. The process of concept creation is such of assigning a new label l_n, $l_n \notin L$, to a subset of X.

Definition D3.5 (Concept elimination)**.** Given a set of concepts L. The process of concept elimination is such of deleting the label l_i, $l_i \in L$, from the set of concepts when it is not longer representative of X.

Concept elimination usually occurs due to temporal evolution. Examples are easily found in the sciences, e. g. "Earth is the center of the universe". Note also that concept creation may emerge from the merge of two concepts or by splitting an old concept into a subset of concepts.

Dynamic systems are described not only by permanent states but also by non-permanent states. These non-permanent states may be further classified as transitory or stationary states. To illustrate this, let us take a blue car as dynamic object.

Example 3.3. A blue car can be described by many characteristics. Among its static characteristic one can name the color, branch, model and year of production. Then, for example, a car can be described by 'blue VW beetle of 1990'. The same car has also non-permanent states as those describing its movement, e.g. moving forward, moving backward, stopped. It is important to emphasize that in the dynamical representation the concept may not be related to the object itself but to its states. For example the concept 'moving' represents a state of the car.

To describe the non-permanent states, characteristics describing the dynamics as the speed or acceleration are necessary. The set of features representing the dynamic information necessary to associate a concept to a non-permanent state of an object are called *dynamic features*. A diagram of this example is shown in Figure 3.2.

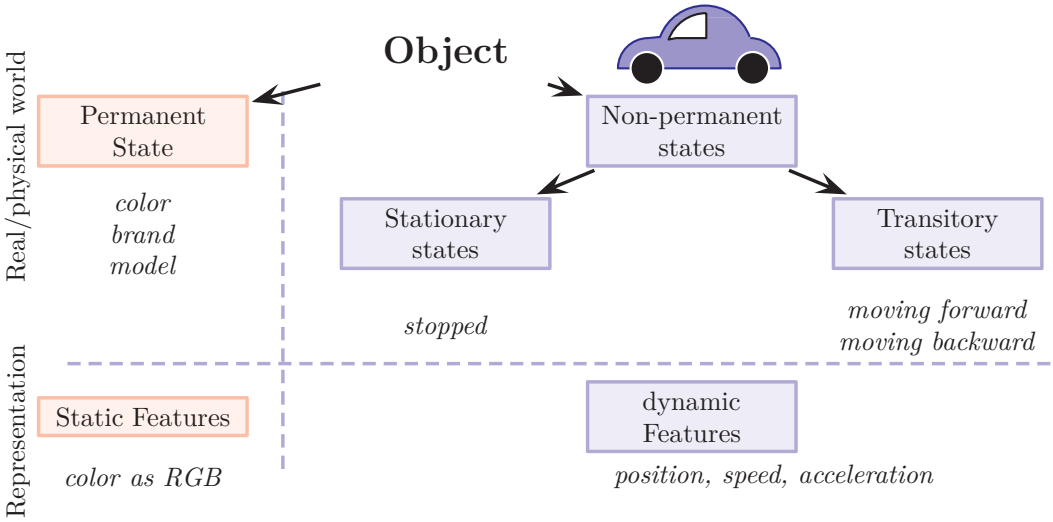


Figure 3.2: Data representation of a dynamic object

It is worth noting that the same characteristic can be used to describe stationary and transitory states. For example the car's velocity can be used to describe stationary states like stop or cruise mode, but also for transitory states as setting off. Nevertheless the features representing this characteristic can be time-independent, as a constant value, or time-dependent, as a mathematical function.

Concepts in the dynamic world may also be subject to concept drift. Let us take the concept 'moving fast'. One century ago cars at $70km/h$ where considered as the fast while nowadays that concept is related with cars at more than $200km/h$. Another example of concept drift can appear regarding spatial information. If a car is circulating in a residential area nowadays at $60km/h$ it is considered as 'too fast' while if it is circulation on the highway, 'too fast' is being over $110km/h$ (or more depending on the country). This example shows that the concept drift is related not only with time evolution but in general with a parameter

evolution.

In fault diagnosis, and in general in pattern recognition, each dimension of the data samples corresponds to a feature carefully chosen to describe at best system's information. If these features are static, they are time-independent and the system is considered as *static*. On the contrary, if a system or a process is described by at least one time-dependent, hence dynamic, feature it is said to be *dynamic*. When features are time-dependent, more information about the evolution of the dynamic system can be extracted by analyzing the feature over time, i.e. by analyzing its trajectory. For example, speed information can be retrieved by analyzing the position of an object over time. Acceleration information can be similarly retrieved from the speed trajectory and so on. This analysis can be used to generate relevant features.

Example 3.4. To better illustrate how dynamic characteristic can be described by dynamic features a three tanks system as the one shown in Figure 3.3 can be used. Lets assume that this system has level sensors in each of the tanks. When the system has no input, the level in all tanks is constant. In this case the tank object can be represented considering the tank level (h_i) at some time point, e.g. $t = 0$. Figure 3.4 shows the data sample representing the tank objects at $t = 0$, in green h_1, in magenta h_2 and in gray h_3. This representation misses the information of the level remaining constant. Now consider that the system input is activated at $t = 5000$, that is, the pump in Figure 3.5 is opened, the level of the tanks starts changing. The lack of information become clearer if the only representation of the system at $t = 10000$ is the one shown in Figure 3.6, where the evolution is not captured.

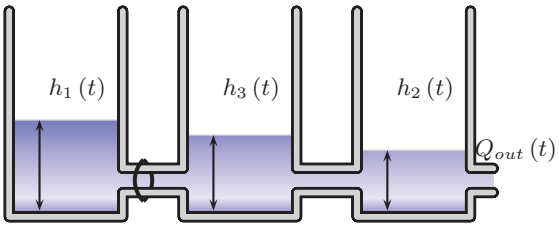


Figure 3.3: Three tank system at $t = 0$ s

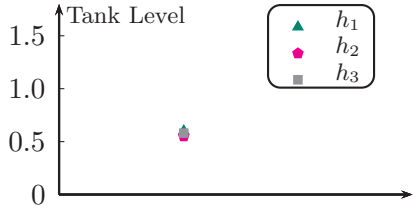


Figure 3.4: Tanks data samples at $t = 0$ s

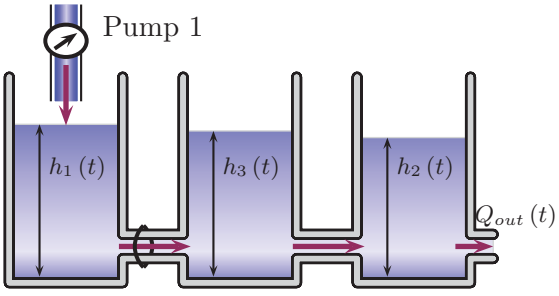


Figure 3.5: Three tank system at t =10000 s

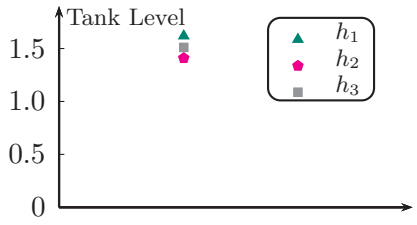


Figure 3.6: Tanks data samples at t=10000 s

If a dynamic system is observed over time, each varying feature is a time-dependent function [Ang 01]. Intuitively, one can imagine that a dynamic system can not be described with the same tools as a static system. In our three tanks example it is interesting to analyze the whole trajectories of the tanks levels as illustrated in Figure 3.7, from where additional dynamic features, like speed, tendency, etc. can also be generated.

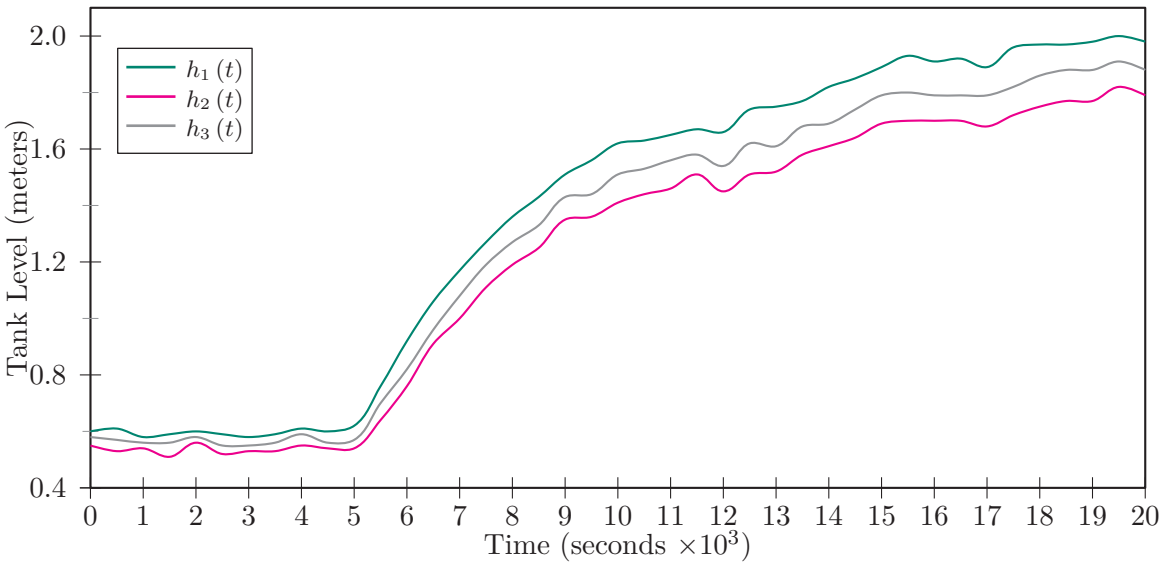


Figure 3.7: Tanks data samples for $0 \leq t \leq 20000$ seconds

The object features are usually presented in the form of a vector known as sample or more precisely data sample. Mathematical definitions for data sample and data set are given bellow, being $\mathbb{F}$ the definition domain of features, which in the case of quantitative features is $\mathbb{R}$ or a subset of $\mathbb{R}$.

Definition D3.6 (Data sample)**.** A data sample is a multi-dimensional representation of an ***object permanent state*** provided by d features. It is denoted by $x_i = [x_i^1, x_i^2, \cdots, x_i^d]$ with $x_i^j \in \mathbb{F}\ \forall\ 1 \leq j \leq d,\ i \in \mathbb{N}$.

Definition D3.7 (Data set)**.** A data set is a collection of N data samples. It is denoted by $X = [x_1, x_2, \cdots, x_N]$.

As established in definition D3.6, permanent states are described by a vector of d static features. Non-permanent states are described by dynamic features whose generation may require the history of the directly measurable features, i.e. their trajectories.

Formally, let x_i^j be a static feature and $x_i^j(t)$ denote a dynamic feature. Dynamic objects having one or more dynamic features are described using dynamic data samples.

Definition D3.8 (Dynamic data sample)**.** A dynamic data sample of a dynamic object is a tuple containing an d_1-dimensional vector of dynamic features and a d_2-dimensional vector

of static features. It is denoted by $x_i(t) = \left(\left[x_i^1(t), \cdots, x_i^{d_1}(t)\right], \left[x_i^1, \cdots, x_i^{d_2}\right]\right)$ with $x_i^j \in \mathbb{F}$ $\forall j : 1 \leq j \leq d_2$ (static features) and $x_i^{j'}(t) \in \mathbb{F}$ $\forall j' : 1 \leq j' \leq d_1$ (dynamic features), subject to $d_1 \geq 1$, $d_2 \geq 0$ and $d_1 + d_2 = d$.

Data samples can arrive as a whole or in stream. If the concept of time is considered the order of arrival of the samples is also important, hence two more concepts have to be defined:

Definition D3.9 (Data stream)**.** A data stream is a real-time, continuous flow of data samples arriving in an ordered manner

Definition D3.10 (Time Stamp)**.** Being x_i a data sample part of a data stream, the time stamp ts_i denotes its arrival time.

3.2 Data classification general framework

The problem of data classification exists in a wide variety of domains. Classification techniques try to learn the relationship between a set of features and a target variable of interest [Agg 14]. This target variable of interest represents a concept as defined in Section 3.1. Formally, the problem of classification has been stated as:

Definition D3.11 (Data classification)**.** Given a data set X and a set of k different discrete values L indexed by $1 \cdots k$, each representing a label, data classification is the task of assigning a label (or its index) to each data sample in X.

Definition D3.12 (Classifier)**.** An algorithm that implements data classification, is known as a classifier.

Ideally, a classifier is designed to fulfill two main targets. First, within a class, objects should be as similar as possible (homogeneity) and second, from one class to another, objects should be clearly distinguishable (heterogeneity). These properties are relative and only depend on the selected features describing the classes.

A classifier structure is defined by the number of classes it outputs and the shape of these classes. Each class is defined by a set of parameters providing the class information in the features space. Depending on the information available for the classifier design, data classification can be done in an automatic way (no prior given structure) or by selecting a given structure and the learning paradigm such as supervised learning, unsupervised learning and reinforcement learning. The goal of supervised learning is to find the set of parameters that better describes each class by using a subset of labeled samples called as *training set*.

From this training set the classifier learn the parameters (or set of rules) that allows the classification of new objects into one of the learned classes.

Reinforcement learning, on the other hand, is the problem of learning a specific behavior (or goal) through trial-and-error interactions with a dynamic environment [Kae *et al.* 96]. The principal difference between supervised learning and reinforcement learning is that the later must explicitly explore its environment.

In the case of unsupervised learning the goal of classification, also known as clustering, is the partitioning of samples without the aid of a training set. Clustering algorithms group objects into a number of clusters based only on the hidden data structure, evidenced by some similarity or dissimilarity measures. The main challenge of clustering arises if the number of clusters is unknown.

Definition D3.13 (Clusterer)**.** A classifier that implements classification under the framework of unsupervised learning, is called a *clusterer*.

Classification can be further cataloged as static or dynamic. Intuitively, if the knowledge about the data set to be classified is limited, the set of concepts describing it may drift (definition D3.3) or new concepts may emerge (definition D3.4 in the course of time. To follow this evolution the location of clusters and even the number of clusters may change.

Definition D3.14 (Dynamic classifier)**.** A classifier which is able to deal with concept dynamism as new data is processed is called a dynamic classifier. Specifically,

- A classifier which is able to change the parameters describing its classes, is a dynamic parameter classifier.
- A classifier which is able to deal with dynamic cluster structure is a dynamic structure classifier.

Definition D3.15 (Fully Dynamic classifier)**.** A classifier which is able to change both its structure (not being limited to just cluster creation) and its parameters, is called a fully dynamic classifier.

Definition D3.15 implies that a fully dynamic classifier must implement cluster creation, merge, split, elimination and drift in an automatic fashion. The extension of the above definitions in the specific case of unsupervised learning is straightforward: a dynamic clusterer is a clusterer able to deal with concept dynamism and a fully dynamic clusterer is a clusterer able to change both its structure and its parameters automatically to follow the concept evolution.

The two kinds of dynamism involved in this work (with reference to the state of the object and with reference to the cluster structure), are further explained in the next section using

the concepts of dynamic pattern recognition. Finally, it is worth noting that we consider the methods classifying static objects with static structure as *static classifiers*.

3.3 Dynamic objects

Dynamic clustering is concerned with the recognition of evolving concepts for both static and dynamic objects. To better illustrate the difference between static and dynamic objects, and the particular interest of this thesis in the later, an example motivated by [Zim 00] is introduced.

Example 3.5. Consider a set of six objects in $\mathbb{R}^2$ observed over the interval $T = [0, 50]$. A snapshot of the directly observable features at $t = 50$ is shown in Figure 3.8 and a projection of the features trajectories from $t = 0$ to $t = 50$ is shown in Figure 3.9.

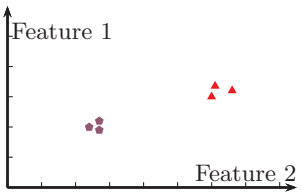


Figure 3.8: States of the objects at t=50

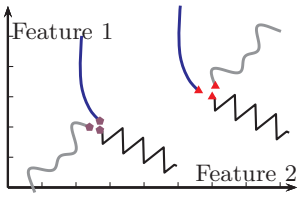


Figure 3.9: Projections of objects trajectories in the feature space

Figure 3.8 shows the point of view that has been used in the classical classification approaches. From this point of view, two classes of objects can be intuitively distinguished in this figure (represented with purple and red colors). However, if one considers the path that has brought the features to that point (the set of their positions over T, as shown in Figure 3.9), the intuition may suggest another result.

The problem of dynamic objects classification can be even more challenging if the dynamic behaviour is heterogeneus, that is, if the parameters describing a feature behavior vary over time. In order to better understand the concept of heterogeneus dynamics, a well known benchmark[2] is used. This data set contains 600 examples of control charts synthetically generated by the process described in [Alc and Man 99]. The control charts exhibit one of five possible dynamic behaviors named cyclic, increasing, decreasing, up shift and down shift. These behaviors can only be noted observing the hole time series. Each subplot in Figure 3.10 exhibits two examples of each one of the listed behaviors in the form of the corresponding time series.

[2]The Synthetic Control Chart time series data available at the UCI machine learning repository [Lic 13]

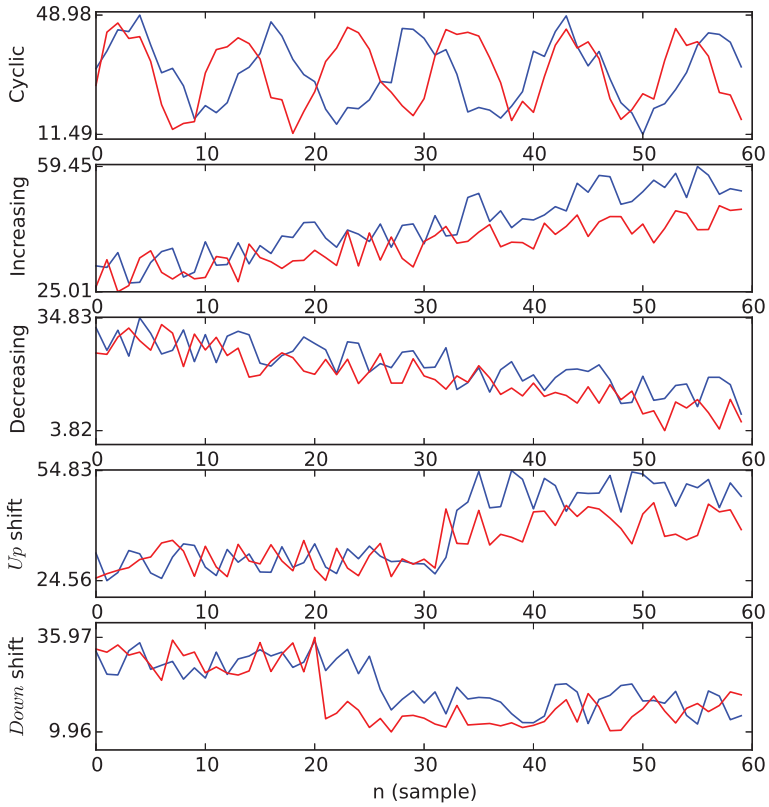


Figure 3.10: Time series exhibit different behavior (cyclic, increasing, decreasing, up shift and down shift)

Example 3.6. Heterogeneous dynamics are illustrated using the blue and red signals in Figure 3.10. Consider the red signals as a set of measures took in a time window T_1 and the blue signals as being the same set of measures but taken in a time window T_2 with $T_1 < T_2$. Each subplot shows these two signals to be slightly different, red signals show original patterns for each specific behavior and blue signals show the evolution of those patterns. The first subplot, for example, shows a reduction in the cyclic signal frequency. The second and third subplots show a change in the slope even if the overall behavior is still the same (increasing/decreasing). The *Up* shift behavior subplot shows a reduction in the shift magnitude. Finally, the last subplot shows that the *Down* shift signal has became smoother, and the abrupt transition between samples $n = 20$ and $n = 21$ has been dragged to samples $n = 27$ and $n = 28$.

The heterogeneous dynamics (changes in patterns) should be taken into account by a detection and diagnosis system. Currently considering these phenomena is quite challenging and research in this field is still very active. The main difficulty lays in feature generation: one has to decide which set of features allows to capture and discriminate the different patterns and how to generate them from the observable characteristics (in our example the time series of Figure 3.10.

As a matter of fact, it has been stated that considering trajectories rather than snapshot values allows earlier detection of malfunctions [Zim 00]. This claim is confirmed from the trajectories from Figure 3.10. If we take the blue trajectories and zoom over the first 20 samples as shown in Figure 3.11, a main drawback of snapshot analysis is evidenced. At $n = 0$ the value of all the features is the same, accordingly, a classifier would classify all the signals as belonging to the same class, even if, analyzing its evolution, it is clear that they exhibit five completely different behaviors. Convincingly analyzing objects dynamically is unavoidable and this fact stresses feature generation as crucial step.

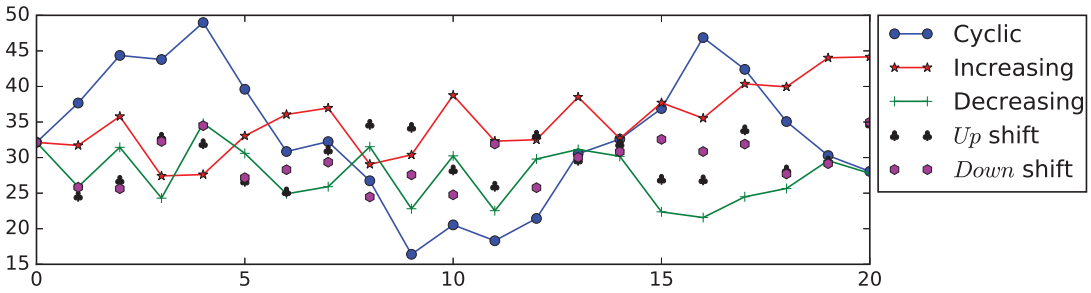


Figure 3.11: Zoom over the first data samples in the Time series

3.4 Dynamic classifiers

Dynamic classifiers exist under different names like, on-line classifier, constructive learning classifier, evolutionary classifier, incremental classifier, evolving classifier, among others. These classifiers exhibit different properties and they may work on different types of objects.

Dynamism in the classifier is achieved when whether the parameters or the classifier structure changes automatically according to new data reflecting system changes. Ideally, abrupt changes in data, also called data shift, are captured as cluster creation or elimination (corresponding to definition D3.4 for concept creation and D3.5 for concept elimination). Smooth changes in data, also called data drift, are usually reflected as cluster drifts (corresponding to concept drift as defined in definition D3.3) and less frequently as cluster merging and splitting. This subsection takes a closer look over the possible changes in dynamic classifiers.

3.4.1 Possible changes in the structure of a dynamic classifier

Creation of clusters If new data cannot be assigned to any of the existing classes, new clusters should be created to represent this novelty behavior. Cluster creation faces the challenge of detecting as fast as possible novel behavior without losing the ability to reject outliers. See Figure 3.12 as an illustrative example.

Figure 3.12: New cluster creation

Left side of Figure 3.12 shows the resulting classification of the data samples arrived from $t = 0$ until $t = t_1$. This result reveals a cluster (purple) and some outliers. Some Δt time later the data distribution has changed, as shown on the right of Figure 3.12. Purple samples have lost opacity indicating that they were assigned to that cluster before the Δt window. No new elements have been assigned to the purple cluster and the zone with high samples concentration is detected as a new cluster (cyan).

Elimination of clusters If no new data is assigned to a cluster and old data has been forgotten the cluster should disappear. Some approaches store the center of forgotten clusters in case of data belonging to that cluster appearing again in the future, allowing faster recognition.

Merge of clusters Two or more clusters may be merged if the region that separates them is filled with new arriving data. In that case the whole region formed by the samples becomes a new integrating cluster. An illustrative example is shown in Figure 3.13, where clusters A and B are merged at time $t + \Delta t$ when the low density area that used to separate them fades into a dense region full of new samples. New arrivals into A cluster are drawn as black squares and new arrivals into B as black triangles. It is interesting to see that at $t = t_1$ the samples between A and B were considered as noise. The evolution of the classifier at $t = t_1 + \Delta t$ allowed data in this area to be not longer considered as noise but as part of the new formed cluster instead.

Figure 3.13: Merging of clusters A and B

Some clustering algorithms merge clusters only in the case of strong correlation between them [Tu and Che 09]. Fuzzy approaches merge the clusters if a large number of new data samples have equally high membership degrees to the clusters [Ang 01].

Splitting clusters A cluster should be splitted into two or more clusters if, with the arrival of new data, high density regions can be distinguished inside the cluster. In that scenario, dense regions are separated by low density regions, making the cluster no longer homogeneous. Even more, the cluster center could be situated in a low density region, loosing its interpretability as prototype of the elements in the cluster. Splitting the cluster creates smaller homogeneous clusters, completely representative of the belonging samples. An illustrative example of this phenomenon is shown in Figure 3.14.

Figure 3.14: Splitting of cluster A

3.4.2 Possible changes in the parameters of a dynamic classifier

Drift of clusters If the data distributions drift slowly over time it is expected that the cluster describing the objects drift accordingly. An example of drift is shown in Figure 3.15 assisted by point opacity, which serves as an indication of the time of arrival of the sample. Older samples are more transparent that the new arriving objects. All clusters are susceptible to drift which may or not cause clusters to become closer and possibly merge.

Figure 3.15: Drift of clusters A and B

3.5 State of the Art: dynamism and classification

An introduction, from a pattern recognition perspective, to objects, both static and dynamic, and to classification algorithms, which can also be considered as static or dynamic, was given in the previous section. This section reviews the classification methods that show dynamism in their structure or parameters and/or that are applied to dynamic features.

These methods are classified based on the nature of the data and the adaptation ability of the classifier. Three main categories are present: 1. Methods classifying static objects using dynamic classifiers, 2. Methods classifying dynamic objects with dynamic parameter classifiers and 3. Methods classifying dynamic objects with dynamic classifiers.

3.5.1 Classifying static objects using dynamic classifiers

Static objects are represented by a set of features that describe their state. Data can come, for example, from sensors or configuration parameters in a process and be of a qualitative or quantitative nature. As introduced before, data can be taken as static either because it does not change in time or because its time related features are not going to be considered. In these cases the classifier follows the changes in the observed data by adjusting its structure to preserve a good performance.

In approaches for classifying static objects using dynamic classifiers the structure and parameters of the classifier can change when different feature arrangements (distributions in the feature space) are detected and no previous knowledge about them is available. That is the case of self-learning adaptive approaches and incremental learning. For example, in the case of fraud detection, the classifier must adapt in order to be able to recognize new forms of fraud, which may arise from a variation of a known fraud or as a completely new fraud. If the data is static it is only in the learning stage that dynamic properties of the classifier would be needed, then any of the methods discussed below classify those data sets.

A common problem that motivates the use of dynamic clustering is that of *novelty detection*. In the supervised framework, novelty detection is the identification of new or unknown data that were not presented to the learning algorithm during training [Mar and Sin 03]. On the contrary, in the unsupervised framework it correspond to the identification of data that were not encountered so far. Dynamic classification techniques of static data have mostly implemented an incremental clustering scheme where only the data arriving in the current time is considered, avoiding to deal with temporal relations or sequential elements. Incremental clustering is unsupervised, self-corrective classification involving incremental learning. The problem of incremental clustering is formulated in definition D3.16.

Definition D3.16 (The problem of incremental clustering)**.** For a given data set X of samples arrived between $t = 0$ and $t = t_1$ and a label set formed with the clustering results till t_1, $L_{(t_1)}$, the problem of incremental clustering is to find the object labels in $L_{(t_2)}$ according to new arriving objects ΔX.

Knowledge coming from the new samples affects the clusterer as new clusters may be created and the existing ones may be modified allowing the system (or knowledge-base) to

evolve over time [Bou 09]. Several examples of incremental clustering algorithms can be found in literature.

Incremental clustering for novelty detection was used in [Kwa *et al.* 15] in order to find faults on semiconductor manufacturing, specifically the method detects whether or not a wafer is normal based on process measurements. In this work only normal wafer data are clustered, the authors argued that faulty data is rare enough to be discarded in order to reduce storage and computational requirements. Normal behavior is represented by several elliptical clusters. When an arriving wafer is found to be normal, the prototype of the closest cluster is updated with the new wafer data using recursive formulae. If the wafer is found to be faulty but it was actually normal a new cluster is created taking this wafer as prototype. Authors argue to be able to detect normal wafers even in severe class distribution skews and to efficiently process massive sensor data in terms of reductions in the required storage.

Incremental approaches have also emerged from the need to handle large amounts of incoming data, arriving at high rates in streaming, which limits the possibility to store all the samples, and reduces the time available for processing the samples. These kind of real time approaches are usually called data stream clustering. One example of this is evolving clustering [Ang and Zho 08]. In such approach an evolving fuzzy rule based algorithm that allows a flexible and evolving system structure is created. One of the main advantages of this method is that it does not need the pre-specification of any parameter. The algorithm includes a gradual evolution in terms of local subsystems as well as in terms of input variables. This gradual simplification is based on a measure of the utility of the rules, in terms of the accumulated time of appearance of the samples that form the cluster that supports that fuzzy rule. The algorithm starts the first cluster from the coordinates of the first data sample. Then, for each incoming data sample the potential (mathematical measure of the data density) is calculated and after that, the potential of the existing cluster centers is updated as well. Comparing the potential of the new data sample with the potential of the previously existing centers, one of the following actions is taken: add a new cluster, remove the cluster, replace the cluster with other formed around the new sample or finally not change the cluster structure [Ang 11].

Different approaches to clustering data streams using sumarized representations of data have been proposed since the 90's. The general structure of stream clustering methods is shown in Figure 3.16. These methods are separated in two components: the first stage makes summarized representations of the arriving data online and the second stage clusters those representations offline. For the online stage tree or grid indexing methods are used to speed-up sample location. For the offline stage two main approaches are found, distance-based methods and density-based methods.

On the side of distance-based methods the CluStream algorithm [Agg *et al.* 03] is one

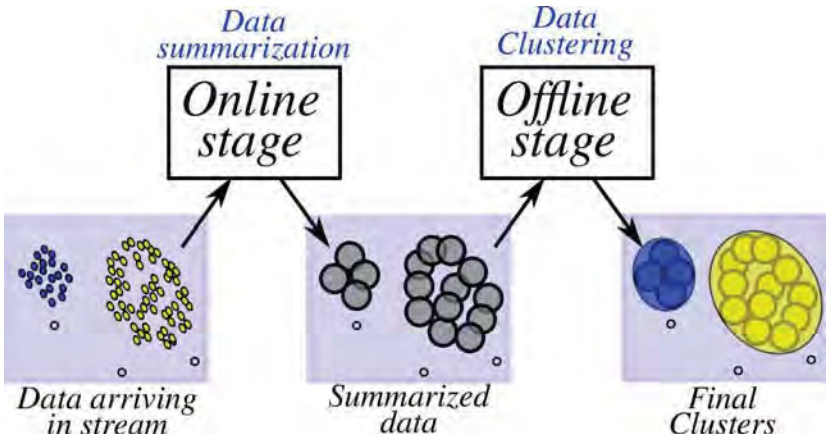


Figure 3.16: General structure of stream clustering methods

of the best known. In its online stage, data are collected, pre-processed and compressed into micro-clusters. In the second stage, the micro-clusters are grouped into clusters (called macro-clusters) for different, specific time horizons, using stored pictures of the micro-clusters (called "snapshots") and a modification of the k-means algorithm. The so-called snapshots allow one to make classifications in different time windows and also to analyse cluster time evolution. This pyramidal time framework, as called in [Agg *et al.* 03], is the main advantage of this algorithm and is used in this thesis. One disadvantage of the technique is the predefined constant number of micro clusters that could lead the algorithm to create several clusters for the outliers, which may leave less space for true clusters. Another disadvantage, derived from the selection of k-means as clustering algorithm, is the inability to find non-convex clusters.

Based on micro clusters as CluStream, ClusTree [Kra *et al.* 11] proposes a hierarchical tree shaped index structure which provides efficient cluster location for sample insertion at different levels of granularity achieving micro-cluster-data assignation even at fast rates. Since the ideal is to locate as many data as possible in the leaf nodes of the tree, the algorithm proposes an "hitch-hiker" approach to help the data which were left in high-level nodes to go down to the leaf level nodes. If a sample y comes before the insertion process of sample x is finished, the algorithm interrupts the insertion process of x and stores it temporarily in a buffer from which it would be taken along in the future as an "hitch-hiker" when a new data arrives looking to be located in a micro-cluster on the same path. ClusTree improves the limitation of CluStream about the fix number of micro-clusters, but establishes a maximum number of micro-clusters and when the algorithm reaches this value, has the same limitations as CluStream. Furthermore, since each cluster can absorb only a limited amount of data, similar samples can be located in different micro-clusters and possibly even in different final clusters. The proposal of a hierarchical index structure saves a lot of time reducing the amount of calculation necessary for micro-cluster location, nevertheless the deep-first descent

approach taken by the authors can lead to non optimal classification, since data could be located within a micro cluster that might not be the closest to it.

Many of the previously named algorithms make, implicitly or explicitly, the assumption that data are generated from a gaussian probability distribution, and, due to this assumption, these algorithms produce convex clusters and cannot deal properly with non-convex data sets. This kind of data sets occur naturally for example in spatial data (e.g. geographic coordinates) and can take arbitrary shapes, as linear, elongated, etc. because of geographical constraints as mountains and rivers [Agg and Red 13].

Density-based methods emerge in response to the need of discovering clusters of arbitrary shape with minimal knowledge requirements. They try to explore the data space in order to find dense regions. The density of any particular sample in the data space is defined either in terms of the number of data samples in a prespecified volume of its locality or in terms of a smoother kernel density estimate [Agg and Red 13]. Density-based clustering methods group toghether samples in high density regions. For instance, two different clusters are assumed to be two high density regions separated by contiguous low density regions [Kri *et al.* 11]. This kind of methods do not require the number of clusters as input parameters.

Among the algorithms that use density-based final clustering DenStream [Cao *et al.* 06] makes a good proposal that handles outliers and cluster evolution with the use of outlier (low density) and potential (intermediate density) micro-clusters. It saves time by searching with priority core (high density) or potential micro-clusters to locate arriving data. The offline stage of this algorithm uses a density-based clustering technique called DBSCAN [Est *et al.* 96] to cluster the micro-clusters. The problem with this algorithm is the high number of user defined parameters, that makes it unsuitable for users with little knowledge of the process. That problem is also seen but to a lesser extent in the DStream algorithm [Che and Tu 07] where context should be provided in order to create a grid fine enough for mapping. An illustration of the grid mapping of DStream is shown in figure 3.17. DStream provides an explicit way to deal with outliers as low density grid areas that are not considered for initial mapping, making the grid location and the posterior cluster formation faster and accurate. A major drawback of using grids is that, with greater dimensionality, density computations become increasingly difficult to perform because of the greater number of cells in the underlying grid structure and the sparsity of the data in the underlying grid [Agg 14].

The data streaming approach has also been used in incremental neural network algorithms. Li *et al.* [Li *et al.* 11] design an online clustering algorithm called Self-Adaptive feed-forward neural network (SAFN), based on a kernel–induced similarity measure. The algorithm is used to learn continuously evolving clusters from non-stationary data [Li *et al.* 11]. In this approach the number of classes can be changed dynamically to create new classes and to drift, merge, split or eliminate the current classes (represented by output layer neurons).

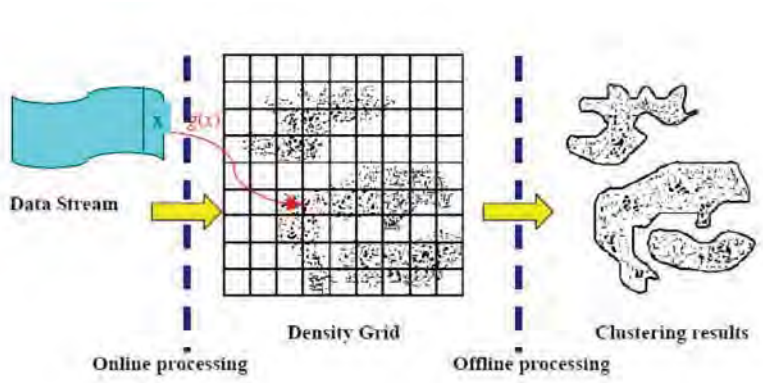


Figure 3.17: Illustration of DStream. Taken from [Che and Tu 07]

This changing structure gives the algorithm the ability for adapting to non-stationary, multi-class data. For unsupervised learning, the learning procedures are built in five main stages one for each of the desirable changes in cluster structure presented in subsection 3.4.1.

Another technique that has the ability of changing its structure in an automatic way is LAMDA (Learning Algorithm for Multivariable Data Analysis) [Agu and Man 82]. This incremental and sequential technique also allows working with quantitative and qualitative data at the same time [Kem *et al.* 06]. The LAMDA algorithm creates (only in learning or self-learning stages) and drift classes in an automatic fashion processing each data sample just one time, avoiding old data to become more important that the current data. LAMDA assigns one sample to one existing class (or creates a new one) by analyzing the contribution of each one of the sample features to the characteristics of the classes. This contribution is called Marginal Adequacy Degree (MAD). Once all the MAD's are calculated, the Global Adequacy Degree (GAD) of the sample to a given class is calculated using fuzzy logic aggregation. This procedure is done for each class. Finally, the sample is assigned to the class with the maximum GAD. LAMDA allows the class feature values to update taking into account the previous data of the class and the values given by the new elements. A diagram of the LAMDA classification method for an sample x_j with n features is shown in Figure 3.18. Kempowsky *et al.* [Kem *et al.* 06] proposed a strategy for the construction of a discrete state machine that enables situation assessment using LAMDA. Isaza *et al.* [Isa *et al.* 09] improved this method to automatically find an optimal space partition, in terms of cluster compactness and separation. This optimization is solely based on the membership matrix of the classification. In [Ora *et al.* 07] the concepts of entropy and information gain are used over the characterized classes to determine the most relevant sensors. The sensor selection in terms of type and location was also adressed in [Hed *et al.* 10].

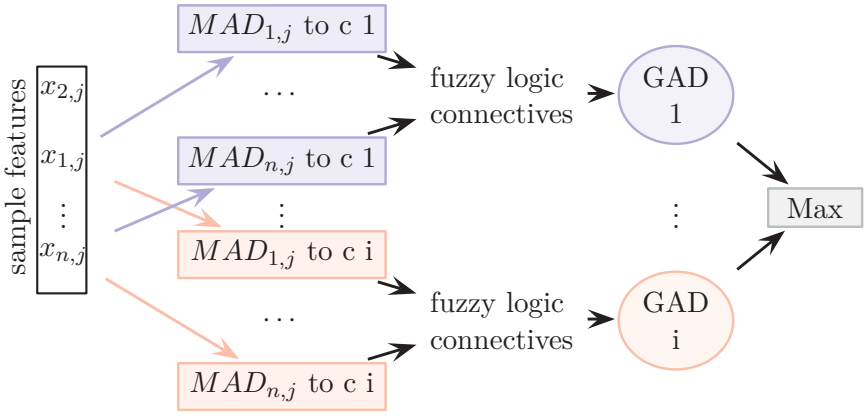


Figure 3.18: General structure of the LAMDA classification method. Edited from [Kem *et al.* 06]

Yet another incremental learning technique is that based on Gaussian Mixture Models (GMM). Considered as a model–based clustering approach, GMM allows to describe a data density function as a weighted sum of Gaussian component densities. Usually trained with the iterative Expectation Maximization (EM) algorithm that requires the storage of the entire data set, the technique was modified in order to work as an incremental learning technique. Two approaches have been used to incremental GMM: refinement–based methods and learning methods. Bouchachia and Vanaret recently proposed an algorithm within the learning methods category that reconcile labeled and unlabeled data. This method enhances the accuracy of the model through Gaussians split and merge operations, based on the volume of the Gaussian and its divergence, respectively [Bou and Van 11].

3.5.2 Classifying dynamic objects using dynamic parameter classifiers

There is a number of applications in which classification of dynamic objects is of primary importance although the cluster structure remains unchanged over time, for example, classification of market characteristics, analysis of customer behavior for marketing or security purposes. For instance, internet user bank transactions are usually monitored based on past user transactions that brings specific patterns. In such cases the user behavior can change in time, influenced by fashionable items perhaps, but the bank possible actions and therefore the classification outputs remains the same (transaction approved, denied or suspicious). If the behavior of the user changes, for example, a personal confirmation must be done in order to ensure the security of the bank account, it is not sufficient to consider only current transaction values to make a correct decision. Dynamic objects can be described by temporal attributes, forming time sets.

An example of dynamic objects can be that of spatio-temporal information coming from GPS technologies or mobile devices. Clustering that kind of data is a critical task in the

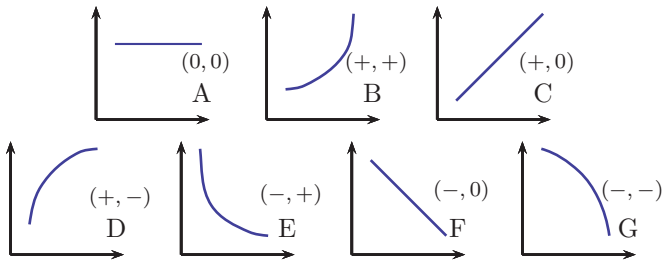


Figure 3.19: Primitives used to represent trends

communications field. Elenkave *et al.* [Eln *et al.* 07] propose a static algorithm to incrementally cluster objects' trajectories in order to recognize evolving groups of moving objects. This algorithm represents a trajectory as a list of minimal bounding boxes and then find the similarity between trajectories as the amount of overlapping between their bounding boxes.

Yet another example comes from industrial environments were a real need to detect tool wear as well as to predict remaining life for effective maintenance is present. A recent example of tool wear detection using static classification techniques can be found in [Kil *et al.* 11]. Kilundu *et al.* propose the use of singular spectrum analysis coupled with a band-pass filter to extract tool wear sensitive features that are then used as input to the classifier.

Trend modelling approaches, also known as Dynamic trend analysis, are a series of techniques to extract relevant information from time series and then use it to draw conclusions about the state of a process. This analysis relies on the fact that similar (or different) events result in qualitatively similar (or different) trends but also that process signals can be represented at different levels of detail. There are mainly two advantages to the qualitative analysis for process trends. The first is the compaction of large amounts of numerical data. The second advantage comes from the fact that a suitable classification and analysis of process trends can detect the underlying class earlier and thus decrease classification time. A formal framework for qualitative representation of process trends was built in the ninety's, and several improvements have been done since. The work of Maurya *et al.* [Mau *et al.* 07], for example, extracts dynamic system trends in an automatic way by means of interval-halving QTA (qualitative trend analysis), and once all possible trajectories are identified and stored in a database, similarity measures are used to select the actual class (using SDG + fuzzy trend–matching QTA). The primitives used in that work to identify the trends are shown in Figure 3.19. Each primitive is identified by a letter assigned according to the signs of the first and second derivatives.

Static approaches have also been used to novelty detection. In those approaches the novelty is detected but the classifier is not updated with the new samples information. Statistical techniques, for example, use statistical properties of the data to achieve novelty detection, ei-

ther by constructing a density function (parametric or non-parametric), by comparing based on distance measures or by using stochastic models (e.g. Hidden Markov Models). The K-nearest neighbors algorithm has also been used in the novelty detection framework. String matching approaches biologically inspired has also been used to differentiate novelties defined as "non-self" elements [Das and Nin 00]. This approaches do not consider the fact that known normal states depend on time, that is, normal behavior evolves in time and values that might be acceptable at a given time might not be in the future. The approaches that consider time variation of the known data and allow adaptation are described in the previous section and in the next. One of the most used clustering approach is the K-means algorithm [Mac 67]. This technique performs a strict partition of samples by means of a simple iterative procedure whose general idea is to classify a set of elements in k groups, k being given in advance. After defining the k centers or prototypes of each group, each element is associated with the group whose center is closer (only one group). Several modifications and improvements have been done to this clustering approach in order to deal with dynamic objects as explained in the next section. Other approaches are: k-nearest neighbors (KNN) algorithm [Sil and Jon 89], that makes no assumption on the statistical properties of the data, instead, the data classification is set as a result of the position of the data sample in relation to its k-nearest samples from the training set. The other techniques as decision trees, discriminant analysis, supervised neural networks, logistic and linear regression can be further consulted in [Tuf 11]. For further information on novelty detection the reader might refer to [Mar and Sin 03].

3.5.3 Classifying dynamic objects using dynamic classifiers

In this category techniques aim to detect and follow dynamic changes on the system behavior by possibly changing the cluster structure. An example of dynamic system that requires dynamic classification is fault detection over long time cycles on industrial applications. To be able to detect a fault multiple objects/systems must be monitored simultaneously. Mechanical pieces may degrade over normal usage, which could lead to failing states. Since degradation does not usually come in short periods of time, only time behavioral analysis can lead to earlier detection. There should also be notice that all systems are susceptible to fall into new states (classes) by means of temporal changes. Yet another example is streaming data classification (video, speech, etc.), because the data to come is not known.

For handling dynamic data three approaches are used: (1) Take only the actual values of the object features, (2) To pre-process trajectories in order to transform them into conventional feature vectors or (3) To define a distance or dissimilarity measure for the trajectories themselves. In either case, the classifier can be updated either using the whole set of samples received from the beginning of observation until the current moment or using only the most

recently received samples (in a window frame).

Approaches based on Current data

Different techniques have been used to classify non-stationary data from a time independent point of view. The development of technology and the need of monitoring processes and systems, has increased the amount of data produced which need further analysis. The analysis of such non stationary, high volume data streams requires high storage capabilities and processing effort. An approach that is becoming stronger among the researchers is that of adaptive incremental learning (presented in section 3.5.1). In cases were in addition to high-volume data real time challenges are added, incremental learning emerge as a feasible solution. This approach can achieve to be computationally efficient for real time applications, with low storage requirements. On the downside, to achieve these goals several dynamic techniques process dynamic data as static with out taking into account temporal information e.g. arriving time or order.

Approaches oriented to pre-processing of trajectories

These approaches take some advantages of the time-related information contained in the behavior of the trajectory but not directly. For example, in the algorithm proposed by Angstenberger [Ang 01], the technique makes advantage of fuzzy set theory to automatically recognize gradual or abrupt changes of the cluster structure in the course of time and to adapt the classifier to these changes. In order to be able to deal with dynamic objects the technique transforms the data sets into a vector of features and then estimates a number of similarity measures for each feature and for each trajectory. This algorithm has been developed for the dynamic unsupervised design of point-prototype-based fuzzy classifiers. The algorithm was applied to a bank customer segmentation problem and to the recognition of typical states in computer network load.

Another example of pre-processing inputs in order to define feature vectors is shown in Filev and Tseng work. They developed a generic evolving model to determine and predict abrupt and gradually developing (incipient) changes for machine health monitoring [Fil and Tse 06]. This algorithm relies on a combination of time and frequency features extracted from time series of measured machine values, process parameters, energy levels, etc. One key feature of this algorithm is that it considers possible multiple operation modes (e.g. start up, normal or idle) as known states and detects the novelty, which in this case is related with faults in the machine. The algorithm is continuously updated implementing an unsupervised recursive learning algorithm. New operating modes can be created and the existing ones are updated. The authors stated two main differences in order to distinguish faulty states from

new operational modes: faults include fewer samples than the normal operating modes and they have limited life.

A dynamic classification approach to novelty detection for process monitoring using class-incremental Fisher Discriminant Analysis (FDA) were proposed by [He *et al.* 09]. They use the normal data set to build the Principal Component Analysis (PCA) model and to identify new faults using the joint plot angle. After a new fault is detected, class-incremental FDA is used to build and update the FDA model (fault signatures matrix) that allows to detect known faults more efficiently than the PCA model, improving the process monitoring performance.

Approaches based on trajectory similarity measures

The similarity measures for time series or trajectories have been studied since the 90's and mainly three kind of methods can be found. Methods based on signature extraction [Wu *et al.* 00, Ber and Hül 06], methods based on data reduction [Meg *et al.* 05, Gam *et al.* 14b] and methods based on temporal alignment [Sak *et al.* 05, Che *et al.* 12]. Signature based methods represent information in a lower dimension space chose to be less costly when computing the distance between signatures. Typical examples of signatures can be found through time to frequency transformations. Feature reduction methods are based on the selection and transformation of features. Examples of transformation can be polynomial representation, piece weighted linear segments or the compression by data point importance. Using sequential temporal alignment, distance between temporal distorted trajectories can be establish. Examples of sequential alignment are the Dynamic Time Warping [Keo and Paz 00] algorithm and longest common subsequence methods [Pat and Dan 94].

The main problem of the approaches based on trajectory similarity measures is that in most cases the method have to be used offline or in batch mode which make them unsuited to high dimensional data or data coming at high rates. Even more, signatures for each behavior must be available beforehand. Existing methods are limited to classify trajectories according to learned signatures and only a few update the class representation as new elements are assigned. Structural changes have been rarely considered and are mostly limited to cluster creation [Ber and Hül 06]. In the best of our knowledge there is no *fully dynamic* method based on trajectory similarity measures.

3.6 Summary

In this chapter a complete overview of data classification have been given with emphasis on dynamism. Section 3.1 presents the main notations and definitions used in this thesis. Then Section 3.2 introduces the framework of data classification emphasizing the difference

between classification and clustering and the concepts of dynamic classifier and fully dynamic classifier.

Section 3.3 introduces the concept of dynamic object and establishes the difference between static and dynamic objects with illustrative examples. These examples show the importance of using dynamic features over static features for monitoring evolving environments.

A state of the art in classification methods considering dynamism is presented in Section 3.5. In this section methods are presented as being part of one of three categories. The first category consider the dynamic classification techniques that deals with static objects (subsection 3.5.1). The second category, presented in subsection 3.5.2, considers the approaches that classify dynamic objects using static classifiers. Finally in subsection 3.5.3 the problem of classifying dynamic objects with dynamic classifiers is presented, showing that, in fact, most of the dynamic classification approaches do not really take into account the dynamism of objects and those approaches considering it do not adapt its structure when new information is available only its parameters. It was seen that most of the dynamic classification techniques make use of the concept of incremental clustering and apply it for novelty detection purposes. Classical approaches deal with dynamic objects by making a static projection of them.

From monitoring to diagnosis

Many researchers have used pattern recognition, neural networks and clustering techniques for fault diagnosis [Kwa *et al.* 15, Isa *et al.* 09, Mau *et al.* 10, Gam *et al.* 14b, Kil *et al.* 11, Hed *et al.* 10]. In general supervised recognition methods work on two stages: training (or learning) and recognition. One known disadvantage of data-driven techniques is that they often address only anticipated fault conditions [Vac *et al.* 07], i.e. they do not consider novelty detection. Other methods use the generation of residuals or the identification of thresholds based on a training set of "tagged" data, in which one or several samples represent one previously measured (normal or faulty) instance.

Classification techniques represent the system functional states by extracting knowledge from various attributes. This knowledge is related to particular behaviors, without being represented by a set of analytical relations. Deviations on this attributes enable the detection of abnormal operations [Isa *et al.* 09]. In classification, when the classes are defined a-priori they can be related to the faulty states directly in the training stage, by means of a fault-pattern library (formed using previously observed faults or expert information) or simply with a knowledge-based interface. On the other hand, in the case of clustering, when the target classes or labels are not know in advance, data can only be clustered in sets, but, with no further interpretable meaning. In such case, the self-learning and the interpretation of the classes in terms of functional states is essential, then a data-reality association must

be done. This association between the classes and the actual situations that they represent can be achieved offline by means of expert intervention [Kem *et al.* 06]. Indeed, some classes have sometimes very similar characteristics and the expert may decide to assign these classes to a single state. In the developing of a supervisory system, the adequate structure must be considered to enable future, medium to long term, reasonable changes. From the desired characteristics for a supervisory system presented in section 2.2, the self-reconfiguration property is by far the less explored. The intelligent supervisory systems should detect, identify and isolate the different possible behaviors, so that the self-reconfiguration response is accurate. In clustering techniques, self-reconfiguration implies updating the classification model online (creating, deleting, merging and splitting of classes), hence the need of fully-dynamic classifiers as explored in this thesis.

Part II

A Dynamic Clustering algorithm for tracking Evolving Environments

In Part I the context and basis to this thesis were presented. This part presents the main contribution of this thesis, a dynamic clustering algorithm that uses distance-based and density-based analyses to cluster data distributions in which the clusters exhibit different levels of density making no assumption on linearity or convexity. The algorithm overview is given in Chapter 4, and then chapters 5 and 6 present in detail the distance-base stage and the density-based stage respectively. A final chapter uses different data sets found in the literature to illustrate the clustering capabilities of the proposed algorithm.

Chapter 4

A Dynamic Clustering algorithm for tracking Evolving Environments

For autonomous systems, it is essential to be able to identify the current state in order to determine the appropriate future action in order to ensure the achievement of the desired goal. In complex systems this identification process is known as situation assessment [Agu *et al.* 12]. If those systems are exposed to evolving environments the situation assessment task become harder. Machine learning techniques have been widely used for tracking the current state of a system [Mil *et al.* 94, Kem *et al.* 06, GA 12, Agu *et al.* 12].

Clustering system data that change in response to evolving environments requires algorithms able to adapt the structure of the clusterer accordingly to data evolution. In this chapter a **Dy**namic **Cl**ustering algorithm for tracking **E**volving **E**nvironments is proposed. This algorithm will be referred to as ***DyClee***. Based on two stages clustering, one based on distance and the other based on density, ***DyClee*** incorporates the advantages of both approaches.

DyClee was developed to work under the unsupervised learning paradigm in an incremental fashion. Many existing learning algorithms do not emulate the way in which humans learn. Humans experience the world on the fly, and carry out incremental learning of sequences of episodes in real time. Often such learning is unsupervised, with the world itself as the teacher [Car and Gro 10]. ***DyClee*** learns in a similar way. It makes no a priori assumption of the data structure but instead it finds it progressively (on the fly), changing the clusterer structure, and hence the concepts representation, to describe the received data.

This distance- and density-based algorithm features several properties like handling non-convex, multi-density clustering with outlier rejection and achieves to be fully dynamic. All these properties are not generally found together in the same algorithm and ***DyClee*** pushes forward the state of the art in this respect.

4.1 Algorithm overview

DyClee uses a two stages clustering approach as those popularized by [Agg *et al.* 03, Cao *et al.* 06, Kra *et al.* 11], among others. In this thesis the first stage clusters data samples into micro-clusters (from now on μ-clusters). These μ-clusters are summarized representations of data objects similar enough to be represented together by using statistical and temporal information. The second stage analyses the distribution of those μ-cluster whose density is considered as medium or high and extracts the final clusters by a density-based approach. The principal assumptions in this stage is that dense μ-clusters that are close enough (connected) are said to belong to the same final cluster and that the system dynamics are stabilizable. Final clusters are also called *natural clusters* in the literature [Kar *et al.* 99b, Agg *et al.* 03, Cao *et al.* 06], and will be referenced as such throughout this document.

Hypothesis H4.1. The dynamics of a system are detectable, that is, they last enough to be detected through system measurements.

Hypothesis H4.2. Similar dynamics are described by statistically similar data.

DyClee input can be static, represented as a multidimensional data set (array form) in which time is not considered, or dynamic if time related features are desired. If the data input is dynamic, information should be presented in the form of a data stream. Unlike other streaming approaches as CluStream [Agg *et al.* 03] and ClusTree [Kra *et al.* 11], that are distance-based, ***DyClee*** incorporates the density of the μ-clusters as a key factor in finding natural clusters, in other words, the μ-clusters density affects data clustering in each of the clustering stages. The way in which density affects each stage will be explained in depth in the following chapters.

DyClee first stage operates at the rate of the data stream and creates the μ-clusters putting together data samples that are close in the sense of the $L1$-norm. These μ-clusters include statistical and temporal information of the data stream. ***DyClee*** second stage takes place once each t_{global} seconds and analyses the distribution of the μ-clusters. The density of a μ-cluster is considered as low, medium or high (cf sections 6.2 and 6.3) and is used to create the final clusters by a density-based approach, i.e. connected dense μ-clusters are said to belong to the same natural cluster.

Both stages work as parallel threads and exchange information to improve each other's performance. The distance- and density-based parallel structure allows the algorithm to find the final clusters even in evolving environments described in high dimensional spaces. This will be shown in the following chapters. The outlier rejection capability is also granted. A global graphical description of ***DyClee*** in block diagrams can be seen in Figure 4.1.

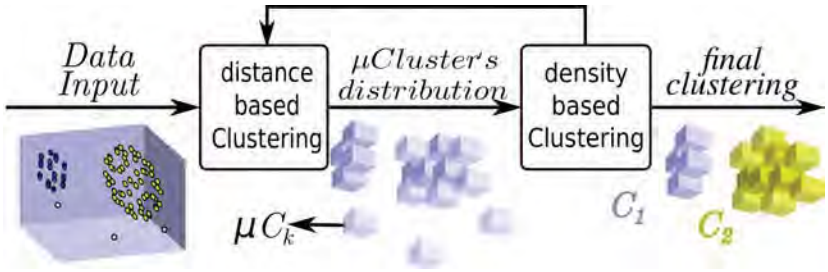


Figure 4.1: Global description of ***DyClee***

4.2 Preliminary definitions

As introduced before, ***DyClee*** takes data samples and clusters them in what we call μ-clusters. As stated in definition D3.6, a data sample is a multi-dimensional representation of an object described by d features. In order to better understand ***DyClee*** structure and operation two preliminary definitions and the corresponding notation are established in this section.

Definition D4.1 (Vector of linear sum LS)**.** Being X a data set of N d-dimensional data samples, $LS\,(X)$ is the d-dimensional vector containing the linear sum of each feature over the N objects in X.

$$LS\,(X) = \left[\sum_{i=1}^{N} x_i^1, \sum_{i=1}^{N} x_i^2, \cdots, \sum_{i=1}^{N} x_i^d\right] \tag{4.1}$$

Definition D4.2 (Vector of sum of squares SS)**.** Being X a set of N d-dimensional data samples, $SS\,(X)$ is the d-dimensional vector containing the sum of squares of each feature over the N objects in X.

$$SS\,(X) = \left[\sum_{i=1}^{N} (x_i^1)^2, \sum_{i=1}^{N} (x_i^2)^2, \cdots, \sum_{i=1}^{N} (x_i^d)^2\right] \tag{4.2}$$

4.3 Algorithm structure

DyClee structure is based on hyperboxes serving as geometrical representations of the set of data samples within. These representations are called μ-clusters. Formally,

Definition D4.3 (μ-cluster)**.** A μ-cluster μC_z is a d-dimensional hyperbox representing the set of n_z data samples that where map to it since its time of creation t_{sz}. Data samples information is summarized in the characteristic tuple $CT_z = (n_z, LS_z, SS_z, t_{lz}, t_{sz}, D_z, lab_z)$.

Being LS_z the linear sum of the elements mapped to the μ-cluster μC_z as defined in D4.1, SS_z the sum of squares as defined in D4.2, t_{lz} the time in which the last data sample was

assigned to the μ-cluster, lab_z the label assigned to the μ-cluster, then D_z is the density of the μ-cluster defined as:

Definition D4.4 (Density of a μ-cluster)**.** The density of the μ-cluster μC_z is denoted D_z and corresponds to the number of data samples mapped to it over its hyper volume V_z.

$$D_z = \frac{n_z}{V_z} \tag{4.3}$$

Since μ-clusters are hyperboxes, their volume is calculated as the product of the size of the box in each dimension $V_z = \prod_{j=1}^{d} S_z^j$, where S_z^j is the size of the μC_z in its jth dimension. ***DyClee*** works over normalized data so the μ-cluster hyperboxes are actually hypercubes with a hypervolume of $V_z = S_z^d$. The size of the hyperboxes S_z^d is set as a fraction of the feature range. This range can be established according to the data context, i.e. minimum and maximum values of the feature, which are used for normalization purposes. If no context is available in advance, it may be established on-line. The hyperbox size per feature is found according to $S_k^d = \phi|max_d - min_d| \ \forall d$, where ϕ is a constant establishing the fraction. For the sake of clarity and without losing generality, from this point onward the superscript ^{d} is removed.

At anytime the μ-cluster center can be found by means of its LS vector. Formally, the center of the μ-cluster μC_z is $c_z = [c_z^1, \ldots, c_z^d]^T$ where $c_z^j = LS_z^j/n_z$.

DyClee structure starts empty and is built by successive sample insertions. The first arrived sample becomes the center of the first μ-cluster, and from then on, the insertion mechanism described next is applied. At the moment of its creation each μ-cluster receives a unique identifier $id_{\mu C_z}$ for which it will be recognized from that time onwards.

DyClee dynamic structure does not guarantee that a μ-cluster μC_z contains all objects which might have been mapped to it, that is, all the data samples within a L_∞ distance $S_z/2$ to μC_z's center. On the contrary, it guarantees that all the objects mapped to μC_z are reachable from it. The definition D4.5 and the Figure 4.2 aim to clarify this property.

Definition D4.5 (Sample to μ-cluster reachability)**.** A data sample x_i is reachable from a μ-cluster μC_z if

$$L_\infty(x_i, c_z) < \frac{S_z}{2} \ \equiv \ \max_j \left| x_i^j - c_z^j \right| < \frac{S_z}{2} \ \forall j$$

In Figure 4.2 two μ-clusters identified in cyan and purple colors contain each several three dimensional samples illustrated with the same colors, i.e. the cyan μ-cluster contains all the cyan samples and the purple μ-cluster all the purple samples. If a sample falls in the intersection of the two μ-cluster, as the red sample in the figure, it is reachable, in the sense of definition D4.5, from both μ-clusters, but it is assigned only to one of them. The next section explains further the insertion mechanism.

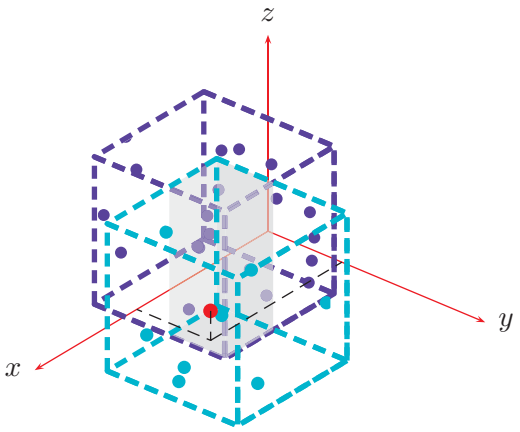


Figure 4.2: The red point is reachable from both μ-clusters.

4.3.1 Insertions

Data samples representing objects are inserted to the most similar μ-cluster from which the data sample is reachable. If there is a tie between several μ-clusters, the data sample is mapped to the denser one. Once found, the structure of this most similar μ-cluster is updated with the data sample information.

In order to speed-up the retrieval of the closest μ-cluster when a new object arrives, μ-clusters are stored in one of two lists. The first list is the 'Active μ-cluster' list (A-list), in this list the μ-clusters to whom the samples are more frequently mapped (medium and high density μ-clusters) are stored. This list will be the first source queried for reachable μ-clusters, giving priority in search to the high and medium density μ-clusters stored in this list over the low density μ-clusters. The second list hence contains the low density μ-clusters, that is, those that have less samples mapped to them. This list is queried only if the data sample cannot be reached by any of the μ-clusters in the first list.

If the data sample is not reachable from the existing clusters in the lists, a new μ-cluster is created with the data sample as center and it is be appended to the 'less-active' μ-cluster list (O-list).

4.3.2 Creation

Clusters represent high density regions in the feature space and consequently, a cluster is created whenever a group (or subgroup) of connected μ-clusters is found to be dense. In other words, when the μ-clusters forming the group (or subgroup) are active μ-clusters. Groups of low density μ-clusters do not form a new cluster. The cluster labeling is done automatically using an incremental numeric index.

4.3.3 Deletions

In order to keep an optimal use of the memory and to reduce computational load, all the μ-clusters no longer considered as representative of the system behavior are deleted. In this thesis μ-clusters are affected with a forgetting factor that makes newer information more important that the information provided by old data samples. Following the same forgetting principle, those μ-clusters receiving no new data, lose density and are eventually pruned when their density falls below a threshold ξ.

4.3.4 Splits

To describe the possible events that might lead to a natural cluster split, the reachability between two clusters must be defined.

Definition D4.6 (μ-cluster to μ-cluster reachability)**.** A μ-cluster μC_z is reachable from a μ-cluster μC_i in the jth dimension if

$$\left|c_i^j - c_z^j\right| < \frac{S_i + S_z}{2}$$

that is, if they overlap in that dimension. Two μ-clusters are said to be reachable if there are reachable in all dimensions, that is:

$$L_\infty(c_i, c_z) < \frac{S_i + S_z}{2} \equiv \max_j \left|c_i^j - c_z^j\right| < \frac{S_i + S_z}{2}$$

In the same way, μC_z and μC_i are said to be unreachable if there are no longer reachable in all dimensions.

A cluster may split into two or more clusters for one of two situation arises:

- μ-clusters composing the cluster are grouped in unreachable subgroups (according to definition D4.6).
- A subset of low density μ-clusters appear between two subsets of high density μ-clusters.

When a cluster splits in two the cluster with the oldest μ-clusters keeps the label of the original cluster and a new label is assigned to the cluster with the newest μ-clusters.

4.3.5 Merges

Two or more clusters can merge if there is an overlapping of the μ-clusters they comprise, that is μ-clusters from both clusters are reachable between them. Specifically, the merge only happens if the reachable μ-clusters have similar densities and they are in the 'Active μ-cluster' list. If different-density μ-clusters groups overlap, no merge is performed. To assign a label

for the cluster formed by the merge process, the labels of the merged clusters are analyzed and the oldest one is chosen as the newly formed cluster label. The new formed group has as additional information the list of old labels.

4.3.6 Drift

Cluster drift is achieved thanks to two different processes reacting at different scales. If the data samples drift slowly enough, arriving samples are prone to be assigned to the same μ-clusters hence updating the μ-clusters center that drifts accordingly. In this situation, the cluster drift thanks to the drift of the μ-clusters composing it. On the contrary, if data drift quickly, arriving samples are probably assigned to new μ-clusters that are created to represent them. In this case the cluster drifts not by the drifting of the old μ-clusters it comprises but by the change in its contents, i.e. the addition of new μ-clusters.

4.4 Summary

We have seen that ***DyClee*** is a dynamic clustering algorithm working in two stages. In the first one samples are grouped in μ-clusters using a distance as dissimilarity measure. In the second stage those μ-clusters are grouped and the natural clusters are found using the denser clusters as components.

DyClee surpasses all the previously mentioned algorithms (*cf.* Section 3.5) in the adaptability of its structure. Unlike them, ***DyClee*** is capable of merging and splitting natural clusters, dealing with concept drift and overcoming problems related to the samples order of arrival.

In fact, as will be seen in chapter 6, ***DyClee*** density-based approach allows local and global density analysis which brings additional advantages to the clustering:

- It can overcome the main problem of most distance-based approaches that can only find convex sets. In ***DyClee*** μ-clusters of similar densities can form natural clusters of any shape and any size.
- Unlike previous density-based approaches as DenStream [Cao *et al.* 06], ***DyClee*** can handle natural clusters exhibiting different density levels.
- Unlike any of existing stream clustering approaches it is capable of detecting of both, local outliers and global outliers.

The following chapters detail each of the algorithm stages, giving special emphasis to the dynamic structure of the algorithm. The distance-based stage is explained in detail in Chapter 5. Then, in Chapter 6, the density-based stage is detailed.

Chapter 5

Distance-based clustering

As introduced in Chapter 4, the first clustering stage seeks to group objects based on the similarity between their features. In order to choose the dissimilarity measure best suited to ***DyClee***, a literature review was performed and is presented in section 5.1 along with a brief analysis of the different measures. This review is summarized in Table 5.1.

To decrease algorithm complexity only one μ-cluster is modified when a new sample arrives. μ-clusters updating process is explained in Section 5.2. This section also explains the forgetting process that is implemented in the algorithm. This process allows the μ-clusters to change in order to follow the evolution of the system.

5.1 Distance dissimilarity measures

As introduced in the first part of this thesis, clustering algorithms aim to group data based on similarity (or dissimilarity). Distance measures have been widely used as dissimilarity measures for classification and clustering algorithms, been the euclidean distance the most popular. Distance measures are used to find the closest cluster (or μ-cluster) to a data sample, but also to find cluster quality measures when inter-cluster and intra-cluster distances are considered. In the following Dis_{iz} represents the distance between a data sample x_i and the cluster center C_z.

Manhattan distance corresponds to the L_1-norm shown in equation (5.1). This measure also called city-block or taxicab distance has been used mostly in clustering problems, especially in high dimensional spaces. Several applications can be found such as Human Expression Recognition [Li and Pen 08], segmentation of multidimensional data [Est *et al.* 01], path based texture segmentation [Fis *et al.* 01], wind speed prediction [Yes *et al.* 13], etc. Variations of this norm can be found in fuzzy ART and ARTMAP [Car *et al.* 92], projected Clustering [Agg *et al.* 99], among others.

$$Dis_{iz} = \sum_{j=1}^{d} \left| x_i^j - C_z^j \right| \tag{5.1}$$

The euclidean norm, presented in equation (5.2), is usually chosen for its geometrical interpretation. The use of this norm creates hyperspherical groups thus ensuring the convexity of the clusters found. This measure has been used by the k-means algorithm [Mac 67] and its fuzzy variant Fuzzy c-means [Bez *et al.* 84]. The Euclidean distance is also used by supervised approaches such as k-nearest neighbours (kNN) [Sil and Jon 89] and most of its variants [Zha and Zho 07, Vik and Jen 14]. In the case of stream clustering, CluStream [Agg *et al.* 03] and ClusTree [Kra *et al.* 11] use this norm to find the distance between the samples and the micro-clusters.

$$Dis_{iz} = \left(\sum_{j=1}^{d} \left(x_i^j - C_z^j \right)^2 \right)^{1/2} \tag{5.2}$$

In many cases the spherical clusters created by the Euclidean norm can be seen as a constraint and ultimately as a problem. Even more, Euclidean distance metrics do not capitalize on any statistical regularities in the data that might be extracted when a large training set of labeled examples is available.

The Minkowsky norm is the generalization of the Manhattan and Euclidean norms, usually called L_p-norm. The Minkowsky distance is shown in equation (5.3). Recent works have shown the possibility of using this norm to improve clustering and data analysis. [Doh *et al.* 04] empirically examines the use of a range of Minkowski norms in the clustering of real world data, showing that fractional L_p-norms (norms with $p < 1$) increase the performance of NN-search. Theoretical validation to the results of [Doh *et al.* 04] can be found in [Fra *et al.* 07]. The fractional version of Minkowsky norm was also used in [Amo and Mir 12], where the authors use different values of p to create a weighted variant of K-means that seems to improve clustering quality in the presence of irrelevant or noisy features. A variant of fuzzy c-means using Minkowsky type norms is found in [Hat *et al.* 00]

$$Dis_{iz} = \left(\sum_{j=1}^{d} \left(x_i^j - C_z^j \right)^p \right)^{1/p} \tag{5.3}$$

Mahalanobis distance (see equation (5.4)) is used in those cases were convex clusters are desired, without the spherical constraint and some statistical knowledge about the distribution is available. This knowledge is represented in the covariance matrix S. Furthermore, most of the existing work in metric learning (from large labeled data sets) relies on learning a Mahalanobis distance or, in other words, learning the S matrix, which exploits possible relations between the features.

$$Dis_{iz} = (x_i - C_z)^T S^{-1} (x_i - C_z) \tag{5.4}$$

From the geometrical point of view, the clusters created based on Mahalanobis distance have hyperellipsoidal forms. Accordingly, a common application of this distance is summarizing input data using hyperellipsoids [Bez *et al.* 84, Ana and Geo 01, Mao and Jai 96]. It is worth noting that, the hyperellipsoid axes are proportional to the eigenvalues of S. Other approaches exploit the relation among the cluster variables focusing on invertible covariance operators [Haa and P¸10, ME *et al.* 14], or finding the best suited S based on an optimisation algorithm [Wei *et al.* 05, Dav *et al.* 07]. In those approaches, Mahalanobis distances usually represent one-class models as only the within-class information is used for their construction.

The sup distance, also known as Chebyshev distance, shown in equation (5.5), makes use of the max norm to measure dissimilarity. An extension of the c-means algorithm using L_1 and L_{inf} norm as pairwise distances [Bob and Bez 91]. [Sou and Car 04] present a clustering method for interval data based on adaptive sup distances.

$$Dis_{iz} = max_j \left(x_i^j - C_z^j \right) \tag{5.5}$$

The cosine distance was used in [Ang and Zho 08] for the online classification of streaming data. The approach, called eClass, is based on a self-developing fuzzy-rule-based classifier system. The cosine distance equation is shown in equation (5.6). The authors claim that this distance can cope with problems, such as a different number of features and zero values. The cosine similarity (second term in equation (5.6)) is one of the most popular similarity measures applied to text documents clustering [Ste *et al.* 00].

$$Dis_{iz} = 1 - \frac{\sum_{j=1}^{d} x_i^j C_z^j}{\sqrt{\sum_{j=1}^{d} (x_i^j)^2 \sum_{j=1}^{d} (C_z^j)^2}} \tag{5.6}$$

The previously review is summarized in table 5.1.

5.1.1 The choice of a meaningful distance measure

Since this thesis aims towards an industrial implementation, the selected distance measure must be capable to work in high dimensional spaces. It is known that in high dimensional spaces the data becomes sparse, and so the performance of distance or similarity measures for data mining applications tends to degrade rapidly as the dimensionality of the data increases. This phenomenon, known as 'The curse of dimensionality', has been extensively studied [Hin *et al.* 00, Rub *et al.* 01, Agg *et al.* 01, Hou *et al.* 10].

The behavior of the L_p norm family ($p > 0$) is studied in [Agg *et al.* 01] showing that the meaningfulness of this measures in high dimensional spaces is sensitive to the value of p.

Norm	Distance	Equation	Principal examples
L_1	Manhathan City-block	$Dis_{iz} = \sum_{j=1}^{d} \left\| x_i^j - C_z^j \right\|$	Fuzzy ARTMAP[Car *et al.* 92] ***DyClee***
L_2	Euclidean	$Dis_{iz} = \left(\sum_{j=1}^{d} \left(x_i^j - C_z^j \right)^2 \right)^{1/2}$	kNN [Sil and Jon 89] k-means [Mac 67] CluStream [Agg *et al.* 03] ClusTree [Kra *et al.* 11]
L_p	Minkowsky	$Dis_{iz} = \left(\sum_{j=1}^{d} \left(x_i^j - C_z^j \right)^p \right)^{1/p}$	k-means var. [Amo and Mir 12] Fuzzy c-means var [Hat *et al.* 00]
	Mahalanobis	$Dis_{iz} = (x_i - c_k)^T S^{-1} (x_i - c_k)$	HEC [Mao and Jai 96] LMNN [Wei *et al.* 05] GKmeans [Gus and Kes 78]
L_{inf}	Sup distance	$Dis_{iz} = max_j \left(x_i^j - C_z^j \right)$	c-means$_\infty$ [Bob and Bez 91]
	Cosine	$Dis_{iz} = 1 - \frac{\sum_{j=1}^{d} x_i^j C_z^j}{\sqrt{\sum_{j=1}^{d} (x_i^j)^2 \sum_{j=1}^{d} (C_z^j)^2}}$	eClass [Ang and Zho 08]

Table 5.1: Distance measures commonly used to establish dissimilarity in clustering

They show that small values of p provide more meaningful results both from the theoretical and the empirical perspective under the assumption of single data distribution. The effects of 'the curse of dimensionality' under the assumption of mixture models data (as opposed to single distribution) were studied in [Hou *et al.* 10].

In the ideal scenario of a data set composed of many well-separated natural clusters, each following their own distribution, a distance measure is meaningful if the query point is guaranteed to land in or very near to one of these clusters, as briefly noted in [Bey *et al.* 99]. Similar results are found in [Ben *et al.* 99], where the authors demonstrate that nearest-neighbor queries are meaningful, both theoretically and practically, if for every cluster, the between cluster distance dominates the within cluster distance.

Theorethical studies performed in [Hin *et al.* 00] show that the L_1 distance is the only L_p metric (with $p \in \mathbb{N}$) for which the absolute difference between nearest and farthest point in a neighborhood increases with the dimensionality.

Given the results of [Hin *et al.* 00, Rub *et al.* 01, Agg *et al.* 01, Hou *et al.* 10, Ben *et al.* 99] proving that the meaningfulness of the L_p norm worsens faster with increasing dimensionality for higher values of p, the Manhattan distance (L_1 norm) was chosen as dissimilarity measure in ***DyClee***. In the classification context, this norm outperforms all the other norms of the L_p family as shown in [Rub *et al.* 01].

The μ-clusters are then naturally shaped as d-dimensional boxes by the use of $L1$-norm

as distance measure. As mentioned in the previous chapter the size of the hyperboxes S_k is set as a fraction of the feature range. If data is already normalized the size of the μ-clusters per dimension is $S_k = \phi$ [1]. The ϕ parameter is the only mandatory user-defined parameter for ***DyClee*** and is tuned experimentally. Even if ϕ can take values from 0 to 1, it is shown in the experimental phase of this thesis (chapters 7 and 10) that the optimal value for ϕ varies only between 0.03 and 0.1 for unnoised distributions and goes up to 0.3 for noised data distributions in the performed tests.

In summary, as stated in the algorithm overview presented in section 4.1, whenever an object x arrives, ***DyClee*** assigns it to the most similar μ-cluster. L_1-norm is used to determine which is the most similar μ-cluster. Once found, the point to cluster reachability property is verified to asses whether or not the object fits inside the μ-cluster. If there is a fit the μ-cluster is updated following the procedure described in the next section.

5.2 Updating the μ-clusters

As stated in Chapter 4, if an object x_i is reachable from its closest μ-cluster μC_z, the μ-cluster tuple CT_z is updated with the object information. Among the attributes of the CT, those subject to changes by the insertion of x_i are:

- the number of data samples mapped into the μ-cluster n_z,
- the vector containing the linear sum of each feature LS_z,
- the vector containing the sum of squares of each feature SS_z,
- the μ-cluster density D_z and
- the time of last update t_{lz}

The two attributes in the tuple that are not modified by the insertion of an object are: μ-cluster time of creation t_{sz} and μ-cluster class $Class_z$. In the non-supervised approach the $Class_z$ attribute is modified only in the second stage of the algorithm when the natural clusters are found. The attributes subject to change are updated with the information of x_i in the following way:

$$n'_z = n_z + 1 \tag{5.7}$$

$$LS'_z = LS_z + x_i \tag{5.8}$$

$$SS'_z = SS_z + x_i^2 \tag{5.9}$$

[1] In the implementation of ***DyClee*** ϕ is called `portion`.

$$D'_z = \frac{n'_z}{V_z} \tag{5.10}$$

$$t'_{lz} = t_i \tag{5.11}$$

where $'$ denotes the updated attribute. The vector attributes are updated by descriptor, that is, $LS'^j_z = LS^j_z + x^j_i$ and $SS'^j_z = SS^j_z + (x^j_i)^2 \; \forall j \in \{1, \cdots, d\}$. As introduced before, μ-clusters updating process might generate changes in the underlying dynamic cluster structure.

In order to maintain an up-to-date structure prone to follow data evolution, the algorithm sensitivity to changes may be further increased. The chosen strategy for increasing the sensitivity is the inclusion of a forgetting process in the μ-cluster's structure. Forgetting old samples gives more importance to incoming data, hence making it more relevant to the algorithm. This process will be explained below.

5.2.1 Representation of the forgetting process

DyClee implements a forgetting process in order to cope with cluster evolution. Specifically, μ-clusters are weighted with a decay function dependent on the current time t and the last assignation time t_{lk}. This function $f(t, t_{lk})$ emulates a forgetting process. When a new d-dimensional sample x_1 is assigned to a μ-cluster μC_z at t_i, the updatable attributes of the feature vector change as follows:

$$n'_z = n_z f(t, t_{lz}) + 1 \tag{5.12}$$

$$LS'_z = LS_{k,j} f(t, t_{lz}) + x_i \tag{5.13}$$

$$SS'_z = SS_{k,j} f(t, t_{lz}) + x_i^2 \tag{5.14}$$

Finally D_z and t_{lz} are updated as stated in equations (5.10) and (5.11), respectively. In general, if no object is added to a μ-cluster during the time interval $(t, t + \Delta t)$, its characteristic tuple at $t + \Delta t$ can be calculated using the characteristic tuple at time t and a decay function for the weighted parameters as follows:

$$CT_k^{(t+\Delta t)} = f(\Delta t)\, CT_k^{(t)} \tag{5.15}$$

In subsection 5.2.2 several options for $f(t, t_{lk})$ are proposed.

5.2.2 The forgetting functions

As stated above, in order to follow the process dynamic evolution, data in clusters are subject to a forgetting process. Numerous machine learning methods have implemented some kind of forgetting function (also called decay function) to be able to detect or track concepts that

drift or shift over time. In this section several decaying functions that can be used to calculate the forgetting factor are proposed. All these functions are implemented in ***DyClee***.

The simplest forgetting function corresponds to a linear decay as given in equation (5.16). The function slope a could be inversely proportional to the time it takes to the function to go from one to zero, $a = 1/t_{w=0}$. This function with $t_{w=0} = 6000$ is plotted in blue in Figure 5.1. A linear decay has been used above all in biological and physical systems. A linear forgetting function was used in [Koy 00] for concept drift adaptation in clustering.

$$f\left(t, t_{lk}\right) = \begin{cases} 1 - a\left(t - t_{lk}\right) & t - t_{lk} \leq t_{w=0} \\ 0 & t - t_{lk} > t_{w=0} \end{cases} \tag{5.16}$$

If a non-forgetting time range is appended to a linear decay, we get a trapezoidal decay profile. This type of function is used as profile in electronic applications. The trapezoidal decay function is given in equation (5.17) where $t_{\sim f}$ represents the no forgetting time, and $t_{w=0}$ the time when the function reaches zero. This function is represented in red in Figure 5.1. A trapezoidal decay was used in [Ang 01] for dynamic clustering, but not as a time function, instead it was related to monitoring parameters.

$$f\left(t, t_{lk}\right) = \begin{cases} 1 & t - t_{lk} \leq t_{\sim f} \\ \frac{t_{w=0} - (t - t_{lk})}{t_{w=0} - t_{\sim f}} & t_{\sim f} \leq t - t_{lk} \leq t_{w=0} \\ 0 & t - t_{lk} > t_{w=0} \end{cases} \tag{5.17}$$

Fuzzy logic makes use of the so called Z-function. This spline-based function uses only two parameters, the extremes of the sloped portion of the curve $t_{\sim f}$ and $t_{w=0}$. The Z-function is given in equation (5.18) and plotted in black in Figure 5.1.

$$f\left(t, t_{lk}\right) = \begin{cases} 1 & t \leq t_{\sim f} \\ 1 - 2\frac{t - t_{\sim f}}{t_{w=0} - t_{\sim f}} & t_{\sim f} \leq t - t_{lk} \leq \frac{t_{\sim f} + t_{w=0}}{2} \\ 2\frac{t - t_{w=0}}{t_{w=0} - t_{\sim f}} & \frac{t_{\sim f} + t_{w=0}}{2} \leq t - t_{lk} \leq t_{w=0} \\ 0 & t - t_{lk} > t_{w=0} \end{cases} \tag{5.18}$$

Statistical processes use overall an exponential decay function. This function is shown in equation (5.19), where λ_d is a positive rate known as exponential decay constant. The function is plotted in magenta in Figure 5.1. This type of functions is the most widely used to model decay since it has applications in all fields of science. A generalization of exponential decay is shown in equation (5.20). It is the decay function used by [Agg *et al.* 03] and [Kra

et al. 11] and a graphical representation can be found in green in Figure 5.1. The change in the base from e to any value β gives interesting properties. For example, if β is chosen to be $\beta = 2\psi$, then the time at which half of the data is forgotten, is $\frac{1}{\psi\lambda_d}$. This function is known as the half life function and is widely used in biological processes. Simpler versions of (5.20) were used in [Bou 11], in particular setting λ_d to one.

$$f\left(t, t_{lk}\right) = e^{-\lambda_d(t-t_{lk})} \tag{5.19}$$

$$f\left(t, t_{lk}\right) = \beta^{-\lambda_d(t-t_{lk})} \tag{5.20}$$

Although exponential decay is been widely used, several processes do not follow this imminent decay dynamics. Logistic functions are the common alternative when dead zones or saturation make part of the process behavior. The Sigmoid function is a particular case of the logistic function which has been widely used in ecology, statistics, medicine, machine learning, etc. to describe, for example, growing processes. It introduces a dead zone concept, which in our case, is used to describe the time gap during which cluster's data should not be affected by the forgetting process. The sigmoid equation is given in (5.21), where c is the value of the sigmoid midpoint and b is the steepness of the graph. The midpoint can be calculated as $\frac{t_{\sim f}+t_{w=0}}{2}$ and the steepness is inversely proportional to $t_{\sim f} - t_{w=0}$. This signal is plotted in cyan in Figure 5.1.

$$f\left(t, t_{lk}\right) = \frac{1}{1 + e^{-b(t-c)}} \tag{5.21}$$

Figure 5.1 shows the shape for the named functions in the case where $t_{w=0} = 6000$, $t_{\sim f} = 2000$. For simplicity in the figure t_{lk} is set to zero. As previously mentioned, ***DyClee*** implements all the functions shown in equations (5.16) to (5.21), allowing proper adaptation to all kind of process evolutions. In ***DyClee*** the forgetting process impacts clusters density. As explained in the previous chapter, each t_{global} period the density of all μ-clusters is re-calculated. Density change might provoke a structural change in clusters distribution. This phenomenon is explained in detail in the next chapter.

5.3 Operation of the distance-based clustering stage

The previous sections introduce the tools and methods that form the distance-based stage. This section aims to clarify how all this mechanism work together to cluster data samples.

The distance-based stage takes as input data samples coming from text files, system messages or databases. Since ***DyClee*** implements incremental learning, the data acquisition is also done incrementally (see Figure 5.2). This approach diminishes memory requirements

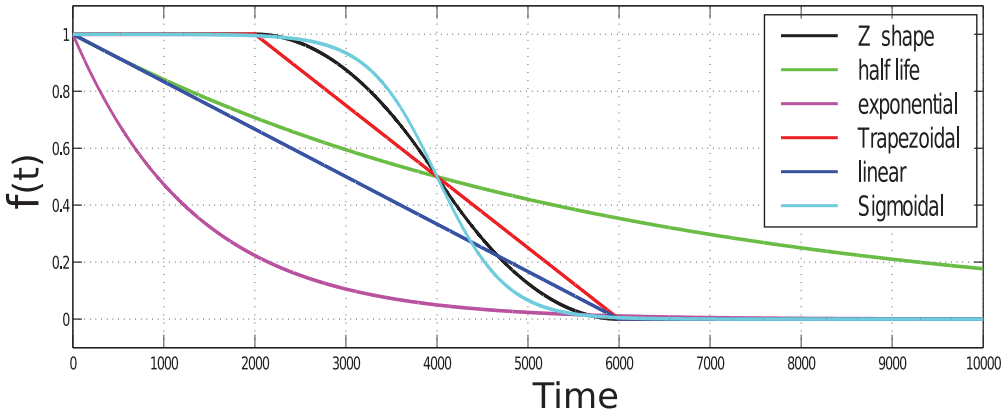


Figure 5.1: Decay functions used to emulate data forgetting

which allows ***DyClee*** to process very large data sets, and even data measurements arriving in stream directly from the process.

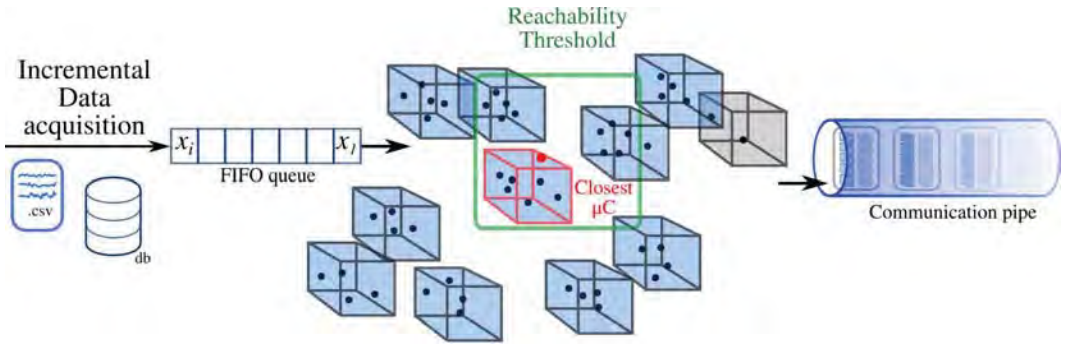


Figure 5.2: Schematic description of the distance-based clustering stage of ***DyClee***

The data 'reader' accesses to data in its original source and charges the data samples into a FIFO queue from which they are processed. For each data sample the set of active μ-clusters from which it is reachable is searched (see green block in Figure 5.2). Among the found μ-clusters a closest μ-cluster query is performed. The found 'closest μ-cluster' is updated with the information of the data sample using equations (5.7) to (5.9) if the forgetting process is disabled or equations (5.12) to (5.14) if it is enabled. The later case includes weighting the μ-cluster CT before adding the information of the data sample. In either case the density and time of last update are updated using equations (5.10) and (5.11) respectively.

If no active μ-cluster is reachable from the data sample, a new reacheability search is perform over the O-list, that is, over the low-density μ-clusters. If the set of low-density μ-clusters from which x is reachable is empty, a new μ-cluster is created using x information as follows:

$$CT = \left(1, [x^1, \cdots, x^d], [(x^1)^2, \cdots, (x^d)^2], curr_t, curr_t, \frac{1}{S^d}, Unclassed\right) \tag{5.22}$$

where $curr_t$ means the current time. Once each t_{global} seconds the A-list and the O-list are sent from this stage to the density-based clustering stage (see Figure 5.2). In ***DyClee*** the distance-based and density-based clustering stages are implemented as separate processes that run in parallel, i. e. using independent memory spaces and CPU resources. These processes exchange information through a bidirectional pipe (see Figure 5.2). The first stage sends the μ-clusters lists as a message to the second stage which is in charge of finding the clusters. This package is sent each t_{global} seconds and after the verification that the previous message has already been processed. Once the classification is made the density-based stage sends to the distance-based stage a message containing:

- The *id* of the μ-clusters that have to be removed from the O-list after being considered as non representative.
- The *id* of the μ-clusters that can be promoted to the A-list.
- The *id* of the μ-clusters to be transferred from the short-term memory to the long term memory.

The next chapter explains how these ids are found as consequence of the density-based analysis. The distance-based clustering stage is summarized in pseudo-code in Figure 5.3.

5.4 Summary

In this chapter the first stage of ***DyClee*** was presented. This distance-based clustering stage summarizes data samples into μ-clusters according to the similarity between their features.

To give a proper context about distance-based clustering some of the most commonly used distance metrics were presented in section 5.1 and an analysis of the significance of this measures in high-dimensional spaces was included. This analysis resulted in the choice of the L_1 norm as the distance measure of this thesis.

In this stage, ***DyClee*** assigns data samples to the closest μ-cluster from which they are reachable. This closest μ-cluster (found using the chosen L_1 norm) is then updated with the data sample information following the procedure stated in section 5.2. In order to increase the reactivity of ***DyClee*** to the changes in the tracked system, a forgetting process was implemented. This forgetting process weights the μ-cluster's CT at each update, using a forgetting function $f(t, t_{lk})$, giving more importance to the arriving records. Six different forgetting functions were proposed and are implemented in ***DyClee***.

Finally, ***DyClee***'s distance-based clustering operation was described in detail in section 5.3 making emphasis in its connection with the second stage of ***DyClee***, the density-based algorithm that will be fully presented in the next chapter.

Algorithm 1: Distance-based clustering stage

```
while x_i in queue do
    if Message in com_pipe then
        com_pipe: receive Message from density-based stage
        Update μC lists with Message
    end
    if no μC exist then
        create μC with x_i info
        append μC to O-list
    else
        Reachables = μC's in A-list that can reach x_i                    // L∞ distance

        if Reachables not null then
            find closest μC in Reachables                                 // Manhattan distance
            update μC with x_i
        else
            Reachables = μC's in O-list that can reach x_i                // L∞ distance

            if Reachables not null then
                find closest μC in Reachables                             // Manhattan distance
                update μC with x_i
            else
                create μC with x_i info
                append μC to O-list
            end
        end
    end
    Write trend to log
    if curr_t − t_last_message >= t_global then
        com_pipe: Send lists to density-based stage
        t_last_message = curr_t
    end
end
close com_pipe
```

Figure 5.3: Pseudo-code of the distance-based clustering stage

Chapter 6

Density-based clustering

In this stage, density-based clustering is executed over the μ-clusters, which allows the algorithm to find clusters of arbitrary shape. A μ-cluster is qualified according to its density as one of three options: dense μ-cluster ($\mathbb{D}\mu$C), semi-dense μ-cluster ($\mathbb{S}\mu$C) or low density (outlier) μ-cluster ($\mathbb{O}\mu$C). The difference between each type is established based on a threshold. $\mathbb{S}\mu$Cs result from an increase in the number of samples assigned to an $\mathbb{O}\mu$C from which one can expect a subsequent cluster creation.

In order to be able to formally define a cluster, two concept have to be introduced: μ-cluster connection and μ-cluster direct connection.

Definition D6.1 (μ-clusters direct connection)**.** Let μC_{k_α} and μC_{k_β} be two μ-clusters, then μC_{k_α} and μC_{k_β} are said to be *directly connected* if they are *reachable* (in the sense of definition D4.6) in all but φ dimensions. The parameter φ establishes the feature selectivity of the classifier and can be set by the user.

Definition D6.2 (μ-clusters connection)**.** A μ-cluster μC_{k_1} is said to be connected to μC_{k_n} if there exists a chain of μ-clusters $\{\mu C_{k_1}, \mu C_{k_2}, \cdots, \mu C_{k_n}\}$ such that μC_{k_i} is directly connected to $\mu C_{k_{i+1}}$ for $i = 1, 2, \cdots, n-1$.

To clarify this concepts a set μ-clusters, represented as squares, are depicted in Figure 6.1. Among the black μ-clusters both direct and indirect connections can be found. An example of direct connection is that between μ-clusters 8 and 7, indicated with the blue arrows. Not all black μ-clusters are directly connected, nevertheless, they are all connected by the direct connections among them. For example μ-clusters 10 and 11, which do not intersect, are connected via μ-clusters 1 and 2. Colored squares do not intersect with any other square meaning that these μ-clusters are isolated.

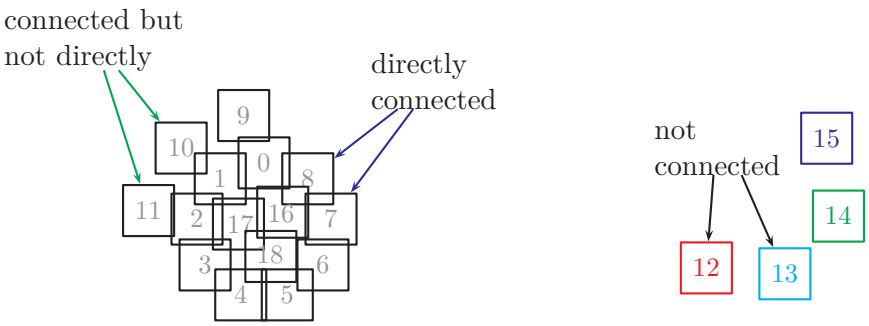


Figure 6.1: Representation of clustering based on μC

Definition D6.3 (Cluster)**.** A cluster is defined as a group of connected μ-clusters where every inside μ-cluster is a $\mathbb{D}\mu$C and every outside μ-cluster is either a $\mathbb{D}\mu$C or a $\mathbb{S}\mu$C. μ-clusters can be connected directly or indirectly.

6.1 Connection search and μ-cluster indexing

The search for all the connections of a μ-cluster is computationally expensive. This task is similar to that of finding the nearest neighbors (NN) which have been widely studied and strategies have emerge to improve this search. The task of finding all the groups of μ-clusters can be seen as that of finding all the existent neighborhoods. In that context, finding the neighborhood of μC_z is the task of finding all its directly connected μ-clusters (neighbors) and then finding the connections of its neighbors and so on. Finding all the μ-clusters directly connected to a μC_z is the same task of finding all the neighbors within S_z distance of μC_z (distance measured using $L_\infty - norm$).

Spatial indexing methods have been widely used to speed-up the NN-query. These methods sort data based on its spatial occupancy. Specifically, they use hierarchical data structures to decompose the data space into regions which can be further decomposed into sub-regions, and so forth. These regions, usually represented as a tree structure, facilitate the NN query and also intersection queries [Fan *et al.* 08].

Since we use hyperboxes to represent data samples the use of a spatial index seems appropriate. Multi-dimensional indexing structures have proved to reduce similarity search time in low to medium dimensional spaces. This indexing should allow fast access to μ-clusters in the first stage and speed-up connectivity search in the second stage. Several options were analyzed in order to find the index best suited to our research problem.

The first approach implemented to speed up the connection search was that of implementing a uniform grid. This approach worked well in low dimensional spaces as reported in [Bar *et al.* 15], but it became computationally more expensive that a brute force search when

the analyzed data set was composed by sparse data samples of more than ten dimensions.

The second approach was to implement a tree-like index structure. Two different tree data distributions were tested in order to find the index structure best suited to our goals. The selected structures are: the 'R*Tree' [Bec *et al.* 90], the 'BallTree' [Omo 89] and the 'KDTree' [Ben 75]. These structures were tested using three data sets exhibiting different spatial behavior: a uniformly distributed data set and two geometrically distributed data sets. Tests were performed varying the number of samples among the values $N = \{10, 100, 1000, 1000\}$ and the number of dimensions as $d = \{2, 4, 8, 16, 32, 64, 128\}$.

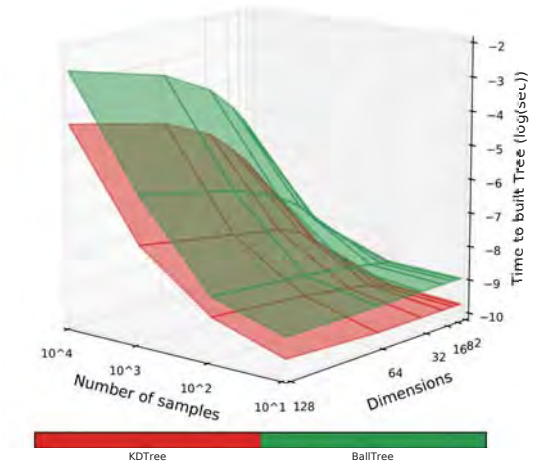
The R*Tree showed the poorest performance in terms of both computation time and memory usage. The BallTree was found to be an order of magnitude slower than the KDTree. The KDTree was selected as spatial index since it proved to be more efficient in μ-clusters group retrieval and index construction. Some graphical results are summarized in Figure 6.2 and the complete test results are shown in appendix A. It can be seen from Figure 6.2 that the tree-like indexes perform poorly when the number of indexed elements is bigger than a thousand. Consequently, in the algorithm implementation a limit of a thousand μ-cluster should be considered if the analyzed dimensions scale up to a hundred.

DyClee implements two different approaches to establish the dense character of the μ-clusters, named global-density approach and local-density approach. The former approach allows to detect clusters with similar densities while the later allows the detection of clusters with varied densities. ***DyClee*** global- and local-density approaches are explained in the following subsections.

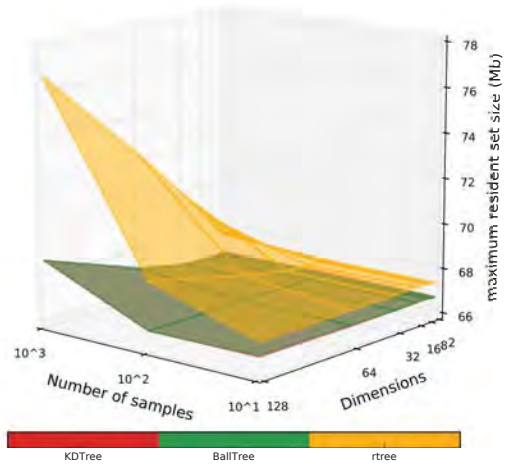
6.2 Global-density approach to clustering

Density-based clustering algorithms group data samples according to density. Exemples can be found in [Est *et al.* 96], proposing the DBSCAN algorithm, and [Cao *et al.* 06], proposing the DenStream algorithm. In these algorithms, the concept of 'dense' is related to a global user-defined parameter: $MinPts$ in the case of [Est *et al.* 96] and $\beta\mu$ in the case of [Cao *et al.* 06].

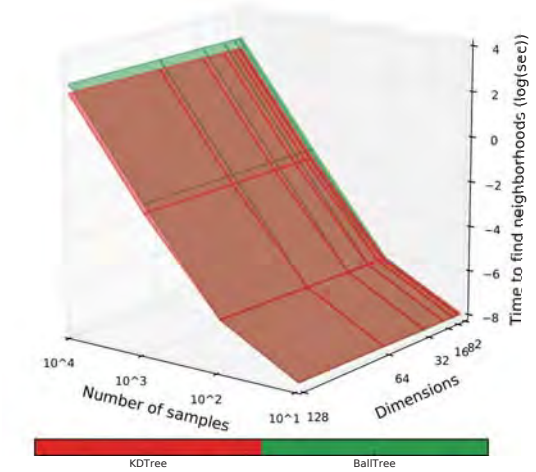
In this thesis' global-density approach, density is considered as a μ-cluster characteristic with respect to all the other μ-clusters. Two measures are considered as representative of theμ-clusters, the average of μ-cluster's density and the median. These measures will work as thresholds for establishing the dense character of a μ-cluster, with no need of any user defined parameter. The intuition behind the selection of these measures is that the median and average densities of an heterogeneous set of μ-clusters are significantly different, although,



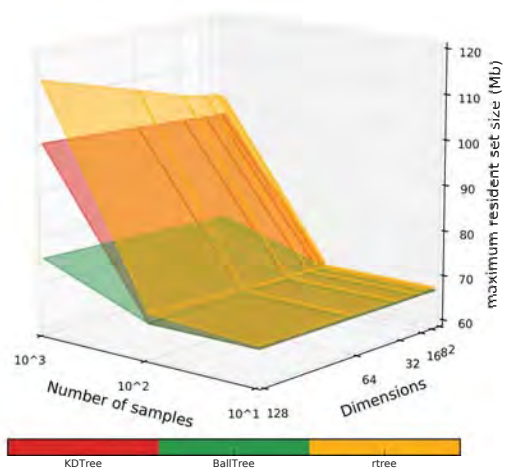
(a) Tree building Computation time ($log_{10}(sec)$)



(b) Tree building Maximum resident set size (Mb)



(c) Group retrieval Computation time ($log_{10}(sec)$)



(d) Group retrieval Maximum resident set size (Mb)

Figure 6.2: Comparison between different tree indexes for the tree building task (Top) and group retrieval task (Bottom) in a spherical distribution. The R*Tree was removed from (a) and (c) to illustrate the difference between KDTree and BallTree

if the set of μ-clusters is uniformly dense, these two quantities are equal.

Formally, a μ-cluster μC_z is said to be dense at time t if its density is greater than or equal to both global measures, i.e. the median and the average. On the contrary if its density is bigger or equal to one of the two measures and lower than the other, the μ-cluster is said to be semi-dense ($\mathbb{S}\mu$C). Finally if the μ-cluster μC_k has a density bellow both thresholds it is said to be an $\mathbb{O}\mu$C. These conditions are represented in inequalities (6.1) to (6.3), where D_i is the density of the μ-cluster i and K is the total number of μ-clusters.

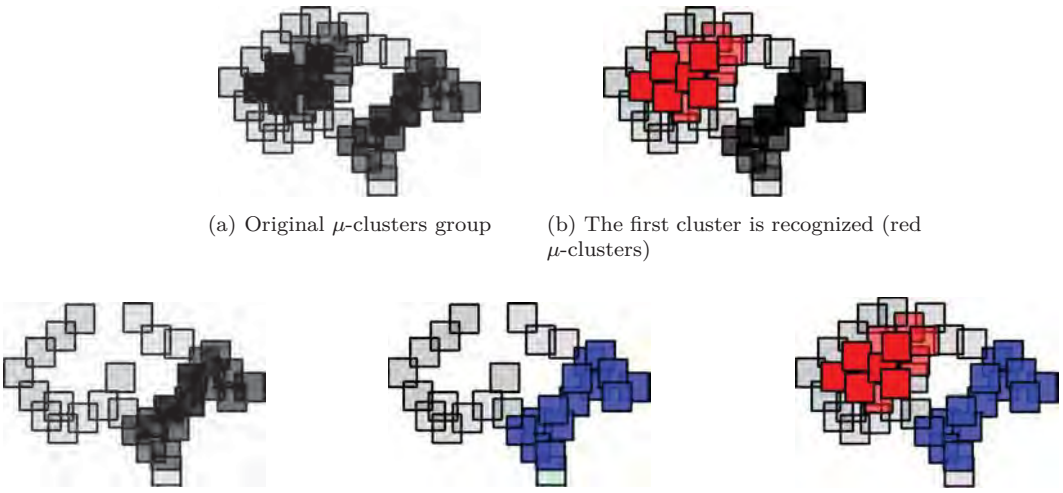
$$\mu C_k \text{dense} \Leftrightarrow D_k \geq median\left(D_1 \cdots D_K\right) \wedge D_k \geq avg\left(D_1 \cdots D_K\right), \tag{6.1}$$

$$\mu C_k \text{semi-dense} \Leftrightarrow D_k \geq median\,(D_1 \cdots D_K) \vee D_k \geq avg\,(D_1 \cdots D_K)\,, \tag{6.2}$$

$$\mu C_k \text{outlier} \Leftrightarrow D_k < median\,(D_1 \cdots D_K) \wedge D_k < avg\,(D_1 \cdots D_K)\,, \tag{6.3}$$

As stated before, in order to find the final clusters, we take into account the dense character of the μ-clusters and their connections. Connections are accounted for by the notion of *group*. Every group of μ-clusters is analyzed one after the other in order to find the clusters. A cluster is created if every inside μ-cluster of the group is a $\mathbb{D}\mu$C and every border μ-cluster is either a $\mathbb{D}\mu$C or a $\mathbb{S}\mu$C.

To better illustrate the cluster formation process let us consider a group of connected μ-clusters exhibiting two areas of high density as shown at the top left of Figure 6.3. The density in the figure is represented as μ-clusters opacity. ***DyClee*** searches the $\mathbb{D}\mu$Cs inside the group and all the μ-cluster around them with medium or high density. This subgroup of denser μ-clusters is qualified as a cluster as depicted in red in Figure 6.3 (Top right). The remaining μ-clusters are then analyzed and, since some $\mathbb{D}\mu$Cs still remain in the group, another cluster is formed with the remaining $\mathbb{D}\mu$Cs and $\mathbb{S}\mu$Cs. This process is shown at the bottom of Figure 6.3. A pseudo-code of the second stage algorithm is shown in Figure 6.4 as algorithm 2.



(a) Original μ-clusters group

(b) The first cluster is recognized (red μ-clusters)

(c) The remaining μ-clusters

(d) The second cluster is identified (blue μ-clusters)

(e) The final clustering

Figure 6.3: Global-density clustering of μ-clusters group. The algorithm start the cluster construction at a dense μ-cluster and include all connected active μ-clusters. Once finished the remaining μ-clusters are analyzed and if with the dense μ-clusters another cluster is formed depicted in blue. Since no more dense clusters are found, the clustering process ends. The final clustering is shown on (e).

6.3 Local-density approach to clustering

The problem of taking a global value to identify a sample as dense appears when clusters with varied densities are present in the same data set as can be seen in Figure 6.5, where

Algorithm 2: clustering using global-density analysis

```
DMC ← μC such that D_k > D_avg ∧ D_k > D_median                    // dense
already_seen= [ ]
for μC in DMC do
    if μC not in already_seen then
        already_seen.append(μC)
        if μC.label=Unclass then
            cid ← get next cluster label
            assign cid as currentμC label
        end
        Connected_μC←all μC in neighborhood overlapped with currentμC
        while Connected_μC not empty do
            neighbor←Connected_μC.pop()
            if neighbor is dense then
                neighbor.label← cid
                already_seen.append(neighbor)
                NewConnected_μCs←all μC overlapped with neighbor in its neighborhood
                for newneighbor in NewConnected_μCs do
                    if newneighbor & is dense then
                        Connected_μC.append(newneighbor)
                    end
                    newneighbor.label← cid
                end
            end
        end
    end
end
```

Figure 6.4: density-based clustering pseudo-code

density is represented by μ-cluster opacity. In this case, density-based algorithms using a global approach may misclassify low density clusters as noise.

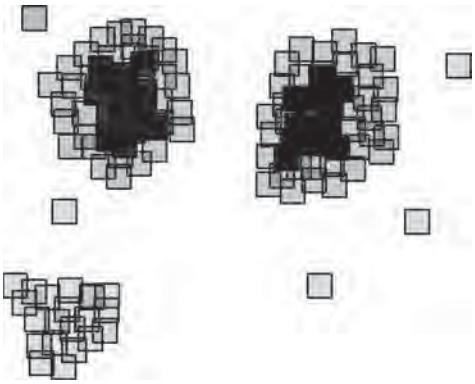


Figure 6.5: μ-clusters groups of varied densities

While it is deemed desirable to detect clusters of multiple densities, it is also important to maintain the ability to reject outliers. ***DyClee***'s solution to multi-density clustering is based on local analysis. Unlike [Est *et al.* 96] and [Cao *et al.* 06], in ***DyClee*** the dense character of each μ-cluster is assessed with respect to the density of the other μ-clusters *in the same*

group. This approach allows what is called multi-density clustering [Mit *et al.* 03].

The average and the median of the μ-clusters within the same group are chosen as thresholds for this group. In other words, for each group G_k, the μ-clusters having their density higher than or equal to the average density of the group ($avg(D_{G_k})$) and higher than or equal to the median density of the group ($median(D_{G_k})$) are considered as dense. μ-clusters having a density higher than or equal to only one of those measures (either average or median) are considered as $\mathbb{S}\mu$Cs and those with density below both measures are considered as $\mathbb{O}\mu$Cs. Summarizing:

$$\mu C_k\text{dense} \Leftrightarrow D_k \geq median\left(D_{G_k}\right) \wedge D_k \geq avg\left(D_{G_k}\right), \tag{6.4}$$

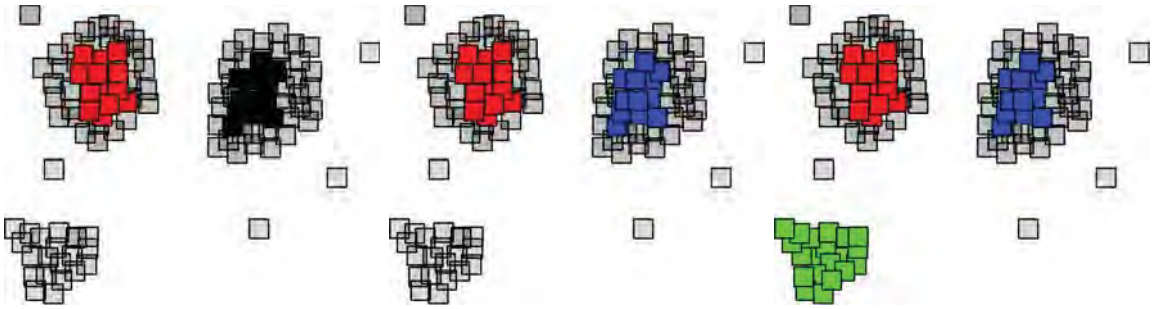
$$\mu C_k\text{semi-dense} \Leftrightarrow D_k \geq median\left(D_{G_k}\right) \vee D_k \geq avg\left(D_{G_k}\right), \tag{6.5}$$

$$\mu C_k\text{outlier} \Leftrightarrow D_k < median\left(D_{G_k}\right) \wedge D_k < avg\left(D_{G_k}\right). \tag{6.6}$$

A cluster is created if every inside μ-cluster of the group is a $\mathbb{D}\mu$C and every border μ-cluster is either a $\mathbb{D}\mu$C or an $\mathbb{S}\mu$C (in a local sense). After this stage, the group G_k is reduced to the μ-clusters that do not contribute to any cluster and the same procedure is applied recursively.

The μ-clusters groups shown in Figure 6.5 are analyzed in this manner: first, for each group, clusters are formed with the denser connected μ-clusters resulting in the two clusters shown in blue and red in Figure 6.6(b). Once formed the rest of the group is analyzed looking for the denser μ-clusters (with respect to the remaining elements of the group). If no dense region is found the next group in analyzed following the same method until all groups are analyzed. The final classification is shown in Figure 6.6(c). The group or groups with the lowest density are taken as outliers. It is worth noting that global-density approaches are unable to detect the green cluster because the densities of the μ-clusters in that group are low with respect to those in the others groups. A pseudo-code of the second stage algorithm is shown in Figure 6.7 as algorithm 3.

In real applications, the collected data samples are generally very unbalanced, having more samples concerning normal modes than faulty modes [Ram and Gou 14]. In ***DyClee*** the use of the local-density analysis allows low density populations (as faults) to be represented as well as high density populations (as is usually the case of normal behavior). In addition, the local-density analysis allows the detection of novelty in its early stages when only a few objects giving evidence of this evolution are present.



(a) First cluster is identified (b) Second cluster is identified (c) Third cluster is identified

Figure 6.6: local-density clustering of μ-clusters groups. (a) the first cluster is identified (red μ-clusters). (b) The second cluster is identified (blue μ-clusters). (c) the low density cluster is identified (green μ-clusters), ending the clustering process.

Algorithm 3: clustering using local-density analysis

```
find Neighborhoods
for neighborhood in Neighborhoods do
    calculate D_avg and D_median from μC in neighborhood
    SMC ← μC such that D_k > D_avg ∨ D_k > D_median                    // dense
    tmp_indx ← ordered(DMC)
    while tmp_indx not empty do
        to_do = { }
        sub_group= { }
        μC ← tmp_ indx.pop()
        while True do
            Connected_μC← all μC in neighborhood overlapped with currentμC
            sub_group.append(μC)
            to_do.append({x ∈Connected_μC |x ∈ sub_group ∧ x ∈ to_do })
            if to_do then
                μC ← tmp_indx.pop()
            end
            break remove from tmp_indx all the elements in Connected_μC
        end
        if ∃x ∈ sub_group|x.label! = 0 then
            analyze group labels()
        end
        cid ← get next cluster label
        assign cid as current sub_group label
        if D_avg of remaining elements in tmp_indx << D_avg of sub_group then
            break
        end
        recalculate D_avg and D_median from the remaining elements in tmp_indx
    end
end
```

Figure 6.7: density-based clustering pseudo-code

6.4 Reactivity to density changes

In ***DyClee*** the forgetting process impacts cluster density. Density change implies μ-cluster type change, its decrease makes $\mathbb{D}\mu$Cs become $\mathbb{S}\mu$Cs and $\mathbb{S}\mu$Cs become $\mathbb{O}\mu$Cs, and vice-versa. When the forgetting process is activated clusters are separated into two groups, active

clusters ($\mathbb{S}\mu$Cs and $\mathbb{D}\mu$Cs) and less active clusters $\mathbb{O}\mu$Cs. A cluster found in the less active list corresponds to one of these two cases: a new $\mathbb{O}\mu$C waiting to grow enough to be representative or an old $\mathbb{S}\mu$C or $\mathbb{D}\mu$C that have lost density due to the lack of new arrivals. The less active list might be seen as a temporary memory that is purged at some frequency. Since active clusters represent the current behavior they should be kept in memory to facilitate access. The memory space where the active clusters are stored will be referred as short-term memory in analogy to the brain storing process.

$\mathbb{D}\mu$Cs are considered a good representation of process behavior at some time point and therefore they should not undergo the forgetting process like the other $\mathbb{S}\mu$Cs or $\mathbb{O}\mu$Cs. Moreover, some processes could go through cyclic states and hence they could correspond to previous states that were defined earlier. Consequently, in order to improve the recognition process, dense characterized behaviors should not be forgotten at the same rate as other behaviors.

The proposed approach to deal with this cyclic or infrequent behaviors properly is to implement a long-term memory along the already used short-term memory. Short-term memory stores the active μ-clusters, that is, those describing the most recent behavior. If the density of a $\mathbb{S}\mu$C drops below a low-density threshold it is tagged as $\mathbb{O}\mu$C and stored into the temporary memory only if it has never been $\mathbb{D}\mu$C before. If it was once $\mathbb{D}\mu$C, it is tagged as $Old\mu$-cluster and stored in a database of old known behaviors (long-term memory). This database is accessed when an input object is not found to belong to any active μ-clusters. This action verifies if the behavior represented by the object belongs to a previously learned behavior, speeding up its recognition in the case where the behavior was already learned before.

Short-term memory is subject to the forgetting process following one of forgetting dynamics explained in Section B.0.3. Long-term memory can follow the same forgetting dynamics that the sort-term memory, with a lower forgetting factor, or can actually be configured to follow completely different dynamics. The configuration of the long-term memory is explained in appendix B.

6.5 Summary

In this chapter two different density-based approaches were presented. In the first, approach the dense character of a μ-cluster is assessed with respect to the density of all the other μ-clusters, these approach is called as global-density analysis. In the second approach, the dense character of a μ-cluster is assessed with respect to the density of the other μ-clusters *in the same group*, these local approach allows multi-density clustering.

Let us notice that single density approaches as DBSCAN [Est *et al.* 96] and DenStream

[Cao *et al.* 06] may have low performance in dissimilar density distributions. In that case, the cluster or clusters with the lower density are considered as noise, missing completely the characterization of those samples. On the other hand, multi-density approaches may fail to reject noise, since they can cluster it in one low density cluster. In order to avoid the inclusion of outliers in multi-density distributions ***DyClee*** mixes local and global density analysis. Local-density analysis is optional in ***DyClee***, and can be set on or off according to the user's needs.

If a sample cannot be assigned to any of the existing μ-cluster, a new $\mathbb{O}\mu$C is formed using the sample's information as model. We assume that μ-clusters with low density ($\mathbb{O}\mu$C) are either outliers or potential clusters in an emerging state. The later case reveals itself with an increment in the cluster density and consequently this μ-cluster grows into a $\mathbb{S}\mu$C as new data is provided as input.

The use of a short and a long-term memory allows two different reactivity rates to system evolution. The first one deals with the most recent behavior stored in the short-term memory. The second deals with those cyclic behaviors that were once representative, but, in the absence of new samples, have lost density and can no longer be considered as active, although they might be revisited later.

Chapter 7

Properties of DyClee applied to static data

The previous chapters presented ***DyClee*** structure based on the fusion of a distance-based summarization stage and a density-based clustering stage. In this chapter ***DyClee*** principles are validated by evaluating its properties using different data sets found in the literature. ***DyClee*** results are also compared to those of several other clustering algorithms. The first section aims at testing ***DyClee***'s capability to capture non-convex structures while dealing with large data sets. Section 7.2 shows ***DyClee*** performance in clustering data sets involving highly non-linear and elongated clusters. Section 7.3 uses a data set of clusters of different sizes bind together by the presence of outliers to illustrate the robustness of ***DyClee*** owing to the use of both distance- and density-based clustering. The ability of clustering in highly overlapped distributions is presented in Section 7.4. As a final experiment, a particularly difficult set of tests is performed mixing together the previous challenges. The results are given in Section 7.5.

For some experiments the following metrics are used to describe the clustering results:

- True Positive: defined for each class as the amount of samples corresponding to the dominant true label with maximum presence in each cluster:

$$T_{p_i} = \max_l (Cl_i \cap lab_l) \tag{7.1}$$

  where Cl_i identifies the cluster i and lab_l is the corresponding true label of the data.

- False Positive: For each cluster the amount of samples assigned to the cluster that do not belong to the dominant true label.

- False Negative: For each cluster the amount of samples of the dominant true label assigned to other clusters.

- Purity: defined as the percent of the total number of objects(data points) that were classified correctly.

$$Purity = \frac{1}{N}\sum_{i=1}^{k} T_{p_i} \tag{7.2}$$

- Precision: Represents the proportion of retrieved samples which are relevant, defined as the number of true positives over the number of true positives plus the number of false positives (F_p)

$$Precision = \sum_{i=1}^{k} \frac{T_{p_i}}{T_{p_i} + F_{p_i}} \tag{7.3}$$

- Recall: Represents the proportion of relevant samples found, defined as the number of true positives over the number of true positives plus the number of false negatives (F_n)

$$Recall = \sum_{i=1}^{k} \frac{T_{p_i}}{T_{p_i} + F_{n_i}} \tag{7.4}$$

DyClee algorithm was implemented using Python programming language (version 3.4). All the details about the implemented functionalities and a brief introduction to the use of ***DyClee*** are provided in Appendix B. The figures of ***DyClee*** clustering results showed in this chapter were also generated using the implemented code.

7.1 Clustering of non-convex sets

When data is arranged in non-convex sets, classic clustering algorithms like k-means and stream algorithms like CluStream and ClusTree tend to fail at clustering them. This section presents ***DyClee*** capabilities to separate neighboring non-convex shaped clusters and to account for local as well as global outliers [Kri *et al.* 09]. To show this property we have chosen to cluster some synthetic sets available in the *scikit-learn* Python module [Ped *et al.* 11]. Our algorithm is compared to scikit-learn module provided implementations of different clustering algorithms, namely MiniBatch KMeans (MBK-m), Agglomerative Clustering, Affinity Propagation, DBSCAN [Est *et al.* 96] and BIRCH [Zha *et al.* 97].

Two noisy data sets named concentric circles and moons were evaluated. These data sets were generated using the scikit-learn data set module. Both distributions, of 1500 samples each, are time invariant. In consequence, ***DyClee***'s forgetting process was disabled for this test.

The clustering results for the concentric circles data set can be seen in Figure 7.1[1]. Clusters are represented using colors, i.e. samples belonging to the same cluster have the same

[1]The algorithm parameters for the test were: MBK-m (# of clusters), Agglomerative Clustering (linkage `"average"`, affinity `"cityblock"`, # of clusters, connectivity (estimated using n_neighbors=10.)), Affinity Propagation (damping=0.9, preference=−200), BIRCH (# of clusters), DBSCAN (eps=0.2), our algorithm (box relative size=0.06).

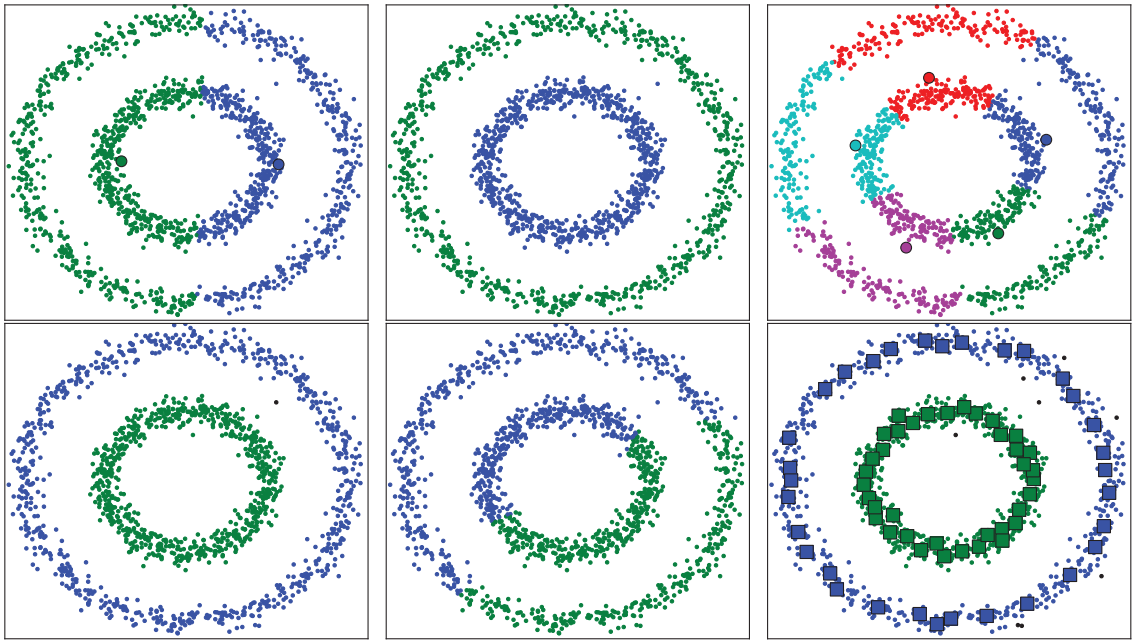


Figure 7.1: Comparison of six algorithms for the concentric circles data set. Top left to right: MBK-m, Agglomerative Clustering, Affinity Propagation. Bottom: DBSCAN, BIRCH, ***DyClee***

color. Detected outliers are colored in black. For illustrative purposes, the cluster centers for the MBK-means and the affinity propagation algorithms are also depicted as colored circles (Figure 7.1 top right and top left). For the same purpose, the $\mathbb{D}\mu$Cs found by ***DyClee*** are depicted as colored squares at the bottom right of Figure 7.1. The agglomerative clustering algorithm, as well as the MBK-m and BIRCH require the number of clusters as initial parameter.

Figure 7.1 shows that MBK-m, BIRCH and Affinity Propagation are not able to cluster the non-convex sets properly, even if the first two know in advance the desired number of clusters. The Affinity Propagation algorithm does not perform well at all in this distribution creating a relatively high number of clusters. Agglomerative Clustering, DBSCAN and ***DyClee*** are able to detect non-convex distribution, nevertheless, Agglomerative Clustering is unable to detect outliers and DBSCAN rejects less outliers than ***DyClee***. Figure 7.2 shows the results over the moons example, for which similar conclusions can be drawn.

7.2 Clustering non-linear elongated clusters

Classical clustering approaches have powerful capabilities in modeling compact data. Nevertheless they mostly fail in detecting elongated structures. For this kind of challenge connectivity based or graph based approaches behave better but they are often very sensitive to outliers. Figure 7.3 shows the three spiral distribution used in [Cha and Yeu 08] to prove

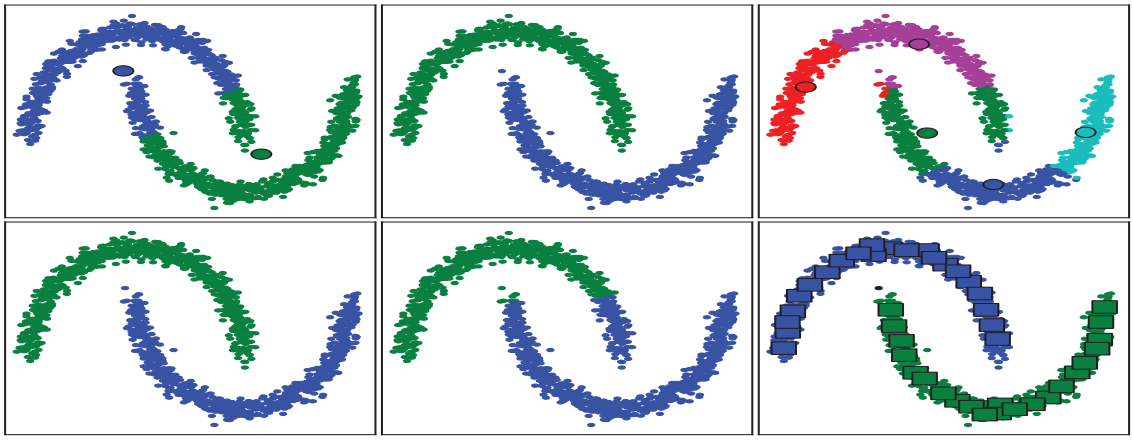


Figure 7.2: Comparison of six algorithms for the moons data set. Top left to right: MBK-m, Agglomerative Clustering, Affinity Propagation. Bottom: DBSCAN, BIRCH, ***DyClee***

what is called as *robust path-based clustering*. In the center of each spiral, samples are more abundant and they become sparser as spiral grows out. This kind of distribution is path based, which makes its clustering specially difficult for an algorithm based just on distance or just on density. Even more, path based clustering methods found in the literature work over the entire data set which make them unsuitable for real-time, large-data applications.

Since ***DyClee*** uses an incremental distance- and density-based approach, it can overcome this kind of challenge, achieving results comparable to those of the original paper by Chang and Yeung [Cha and Yeu 08] using the so-called robust path-based spectral clustering method over this unnoisy set. ***DyClee*** clustering results and the original results of [Cha and Yeu 08] are shown in Figure 7.3. These results are achieved forcing ***DyClee*** to cluster all samples (with the option `Unclass_accepted=False`).

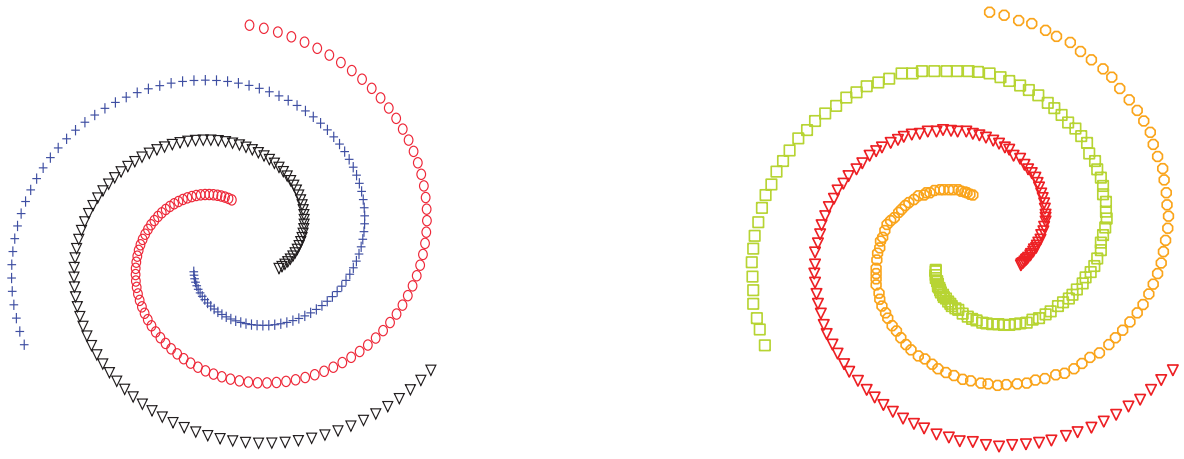


Figure 7.3: Chang and Yeung robust path-based spectral clustering (left) and ***DyClee*** clustering results (right) for the test case in [Cha and Yeu 08].

Robust path-based clustering like [Cha and Yeu 08] gives a measure of the inter-point

Algorithm	Ref	Purity	Precision	Recall	# of Clusters
Path-based	[Cha and Yeu 08]	1.0	1.0	1.0	3
DyClee	[Bar *et al.* 16d]	1.0	1.0	1.0	3
CluStream	[Agg *et al.* 03]	0.43	0.43	0.82	3
DenStream	[Cao *et al.* 06]	1.0	1.0	0.6	32

Table 7.1: Clustering evaluation of streaming methods over the test case proposed in [Cha and Yeu 08]

similarity arguably robust in presence of noise and outliers. The main advantage of ***DyClee*** over path-based methods, and in particular over the robust path based method in [Cha and Yeu 08], is its ability to automatically recognize outliers. ***DyClee*** not only works under a unsupervised paradigm which do not demand a priori knowledge of the number of clusters, but also exclude outlier samples from the clusters using the $\mathbb{O}\mu$Cs. In [Cha and Yeu 08] the noisy concentric circles example used in the previous section was also considered and Robust path-based clustering is unable to correctly label the noisy samples as outliers.

In [Bar *et al.* 16d] ***DyClee*** performance were evaluated against other streaming methods, namely CluStream and DenStream on this data set. Table 7.1 summarized the clustering results of the robust path-based approach and the streaming methods. Table 7.1 shows that ***DyClee*** outperforms the other streaming methods. CluStream put together samples of different clusters which is reflected in its low purity. DenStream is unable to follow the clusters density evolution (clusters are much more dense in the center) which force it to create an elevated number of clusters i.e. 32.

7.3 Clustering aggregation problem

The clustering aggregation problem was presented by Gionis *et al.* in [Gio *et al.* 07]. This problem defined as that of finding a single clustering that agrees as much as possible with the information gathered from other clusterings used as input. Clustering aggregation is claimed to be useful in improving the robustness of clustering. Figure 7.4 shows the test case used by Gionis *et al.* in their clustering aggregation problem [Gio *et al.* 07]. An intuitively good clustering for this data set consists of the seven perceptually distinct groups of samples. The authors of [Gio *et al.* 07] ran five different clustering algorithms implemented in MATLAB (single linkage, complete linkage, average linkage, Ward's clustering, and k-means), setting the number of clusters to 7 in each case. Nevertheless, the correct clusters distribution was not found by any of the named algorithms. Gionis *et al.* [Gio *et al.* 07] then proposed several solutions using clustering aggregation, in particular deriving a clustering distribution that merges the results of the five clustering algorithms used in the previous stage. Unfortunately the aggregation techniques they consider are NP-hard, which makes their implementation not feasible for practical problems.

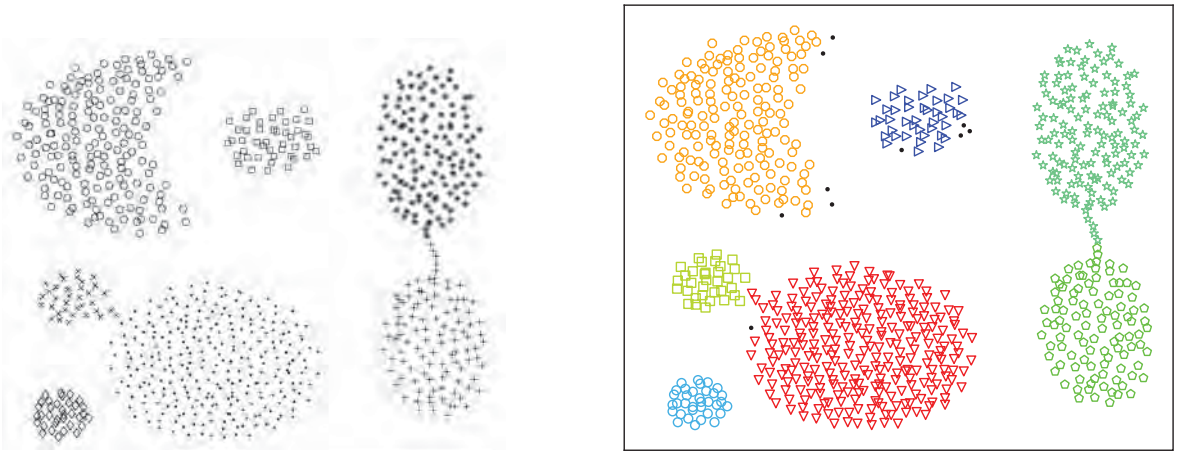


Figure 7.4: Gionis *et al.* clustering aggregation results (left) and ***DyClee*** clustering results for the test case in [Gio *et al.* 07].

DyClee characteristics make ***DyClee*** suitable for the test case presented in [Gio *et al.* 07]. As can be seen in Figure 7.4, ***DyClee*** correctly clusters the test set, achieving almost the same result as Gionis *et al.* whereas Gionis *et al.* solution requires to aggregate the results of five different clustering methods. ***DyClee*** does not even require the number of clusters as input parameter.

Using this data set one interesting aspect of ***DyClee*** can also be exemplified, its ability to reject clusters with size below the generalized tendency. If the clusters distribution is assumed to be relatively uniform, rejecting clusters of small sizes can improve outlier detection. If the ***DyClee*** user selects to enable the minimum size cluster option, `minimum_mc`, clusters with low amount of μ-clusters are not accepted and then taken as noise/outliers. Figure 7.5 shows ***DyClee*** clustering results with this option set to `True`. As can be seen from the figure, this restriction cause clusters of low sparsity to be eliminated and its samples registered as noise (colored black).

7.4 Clustering in highly overlapping situations

In order to illustrate ***DyClee*** performance in presence of high overlapping natural clusters we selected the R15 data set from [Vee *et al.* 02]. This data set of 600 samples, generated by 15 similar 2D Gaussian distributions is shown on the left of Figure 7.6.

DyClee deals with high overlapping data distributions by design. Its implementation of μ-clusters of different densities allows it to detect low density regions between connected cluster which subsequently enables the detection of adjacent clusters.

Figure 7.6 shows ***DyClee*** clustering results compared to those of the original paper [Vee *et al.* 02]. The 15 classes are correctly recognized and outliers are detected. For this test the

`portion` parameter (*cf.* appendix B) was set to 0.02.

This test allows us to show ***DyClee*** sensitivity to changes in the `portion` parameter, that is, the relative size of the μ-clusters. It is worth remembering that `portion` is the only mandatory field for running ***DyClee*** and it can take values from 0 to 1. Figure 7.7 shows ***DyClee*** clustering results when the μ-cluster relative size varies from 0.01 to 0.06 of the feature range. When the μ-cluster relative size is too low, values under 0.015, more clusters are created due to density changes inside the cluster, in other words, regions of low relative density are found between regions of higher density, forming two clusters as can be seen in Figures 7.7(a). On the other hand, if the μ-cluster relative size is taken too large (as in Figure and 7.7(c) and 7.7(d)) clusters can be incorrectly merged due to proximity of μ-clusters of relative high density. For the R15 example ***DyClee*** can correctly classify the data set using relative μ-cluster sizes from 0.015 up to 0.039 as can be seen in Figure 7.7(b). A graphical summary of the number of clusters found when `portion` varies from 0.01 to 0.045 is shown in Figure 7.8.

In some situations clustering all samples might be desirable as can be in the case in fraud detection or quality control. ***DyClee*** allows the user to impose a label to all the samples, i.e. include in the clusters the outliers which are otherwise detected as non representative samples. This option is activated by means of the `Unclass_accepted` parameter. If this optional parameter is set to `False` ***DyClee*** assigns a label to all samples even if they are identify as outliers. For the R15 data set the clustering results obtained when `Unclass_accepted` is set to `False`, i.e. ***DyClee*** is forced to cluster all samples, are shown in figure 7.9.

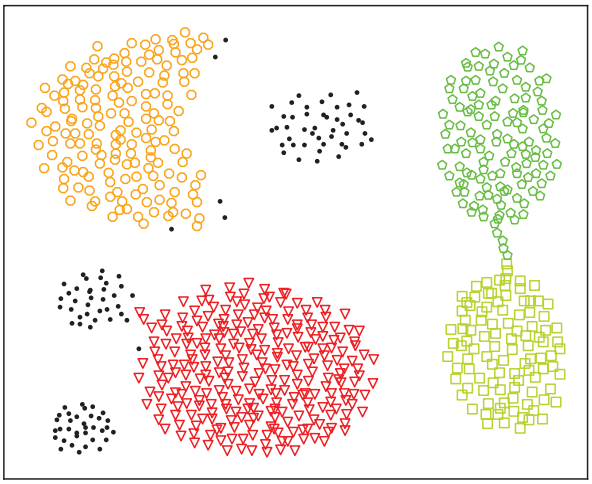


Figure 7.5: ***DyClee*** clustering results for test case in [Gio *et al.* 07] if the option `minimum_mc` is set to `True`.

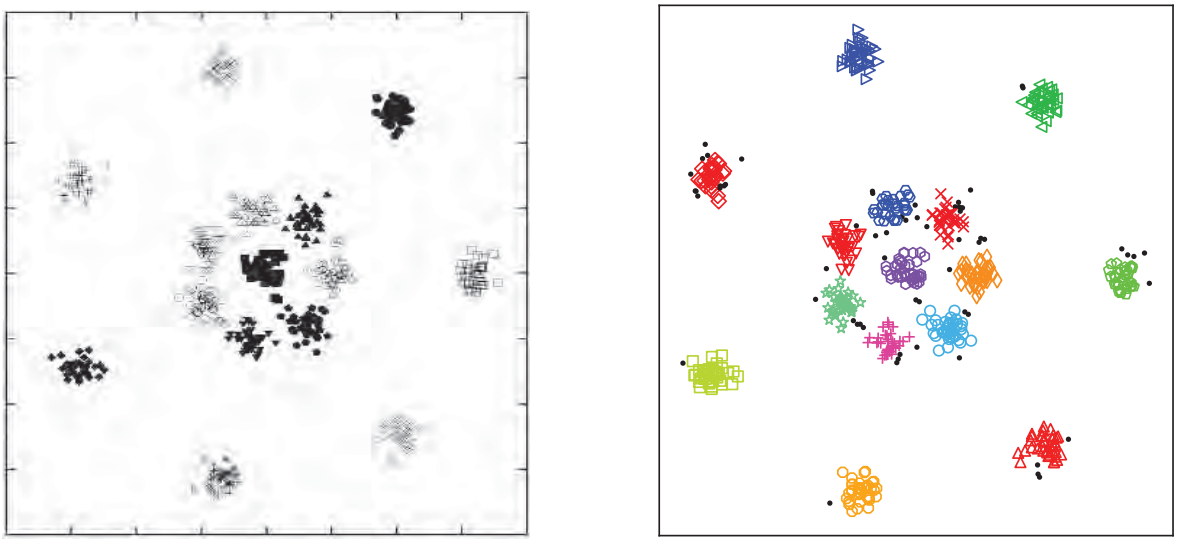
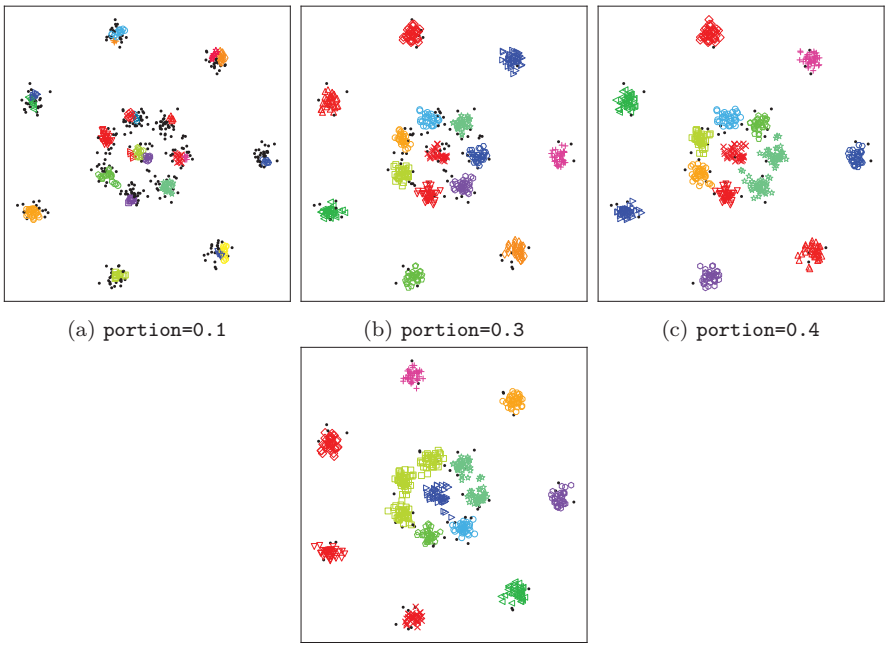


Figure 7.6: Veenman Maximum Variance Cluster Algorithm (left) and ***DyClee*** clustering results (right) for the test case in [Vee *et al.* 02].



(a) `portion=0.1` (b) `portion=0.3` (c) `portion=0.4`

(d) `portion=0.6`

Figure 7.7: ***DyClee*** clustering results. μ-cluster relative size of 0.04(left) and 0.06 (right)

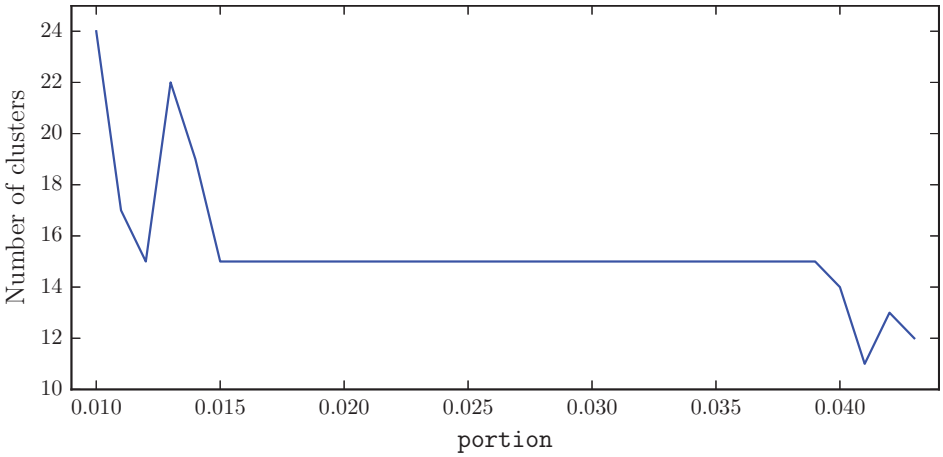


Figure 7.8: ***DyClee*** sensitivity to μ-cluster relative size (`portion` parameter)

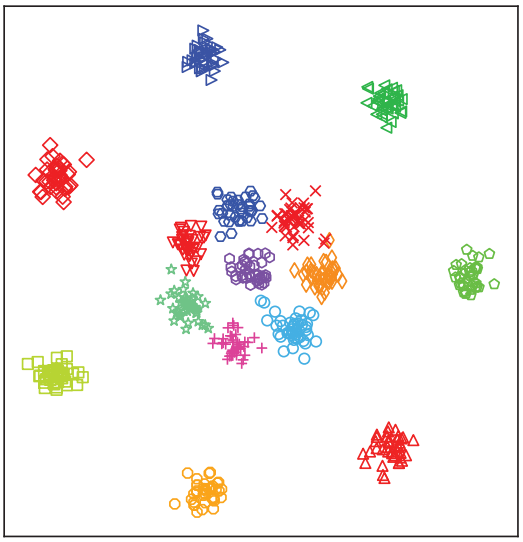


Figure 7.9: ***DyClee*** clustering results. μ-cluster relative size of 0.02 and `Unclass_accepted = False`

7.5 Clustering Chameleon data sets

Finally we consider the $t4.8k$ data set from the Chameleon data sets[2]. Chameleon is a hierarchical clustering algorithm developped by Karypis *et al.* in [Kar *et al.* 99b]. The $t4.8k$ data set has six clusters of different size, shape, and orientation, as well as random noise samples and special artifacts such as streaks running across clusters making its classification particularly difficult. For this data set Chameleon finds eleven clusters, out of which six correspond to the genuine clusters in the data set, and the rest contains outliers. Chameleon clustering results over this data set are the poorest over the five data sets proposed in [Kar *et al.* 99a] and this test case was excluded from the final Chameleon Paper [Kar *et al.* 99b]. Nevertheless, this data set remains an interesting test case for clustering algorithms and

[2] Available at `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download`

has been extensively used for clustering validation [Bor and Bha 07, Fah *et al.* 10, Tra *et al.* 14, Tha *et al.* 15]. The original Chameleon paper which includes this data set is available on the page of the author [Kar *et al.* 99a]. ***DyClee*** results for this test case are given in Figure 7.10, showing the correct detection of only six natural clusters, which correspond to the genuine clusters, but including some outlier samples in the natural clusters. ***DyClee*** results are quite good for this difficult clustering problem.

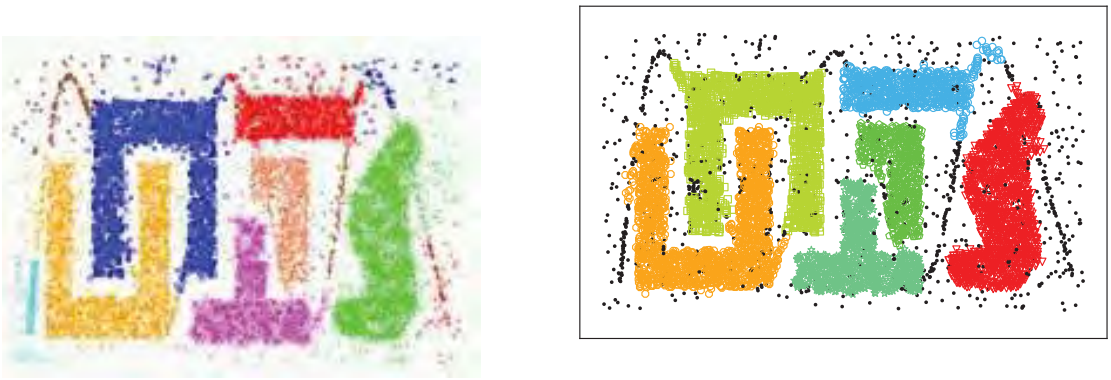


Figure 7.10: Chameleon (left) and ***DyClee*** clustering results (right) for the $t4.8k$ data set in [Kar *et al.* 99a].

7.6 Summary

In this chapter the ability of ***DyClee*** to cluster complex static data sets is tested and it was proven that it is able to capture non-convex structures along with non-linear elongated clusters, an ability infrequently present in distance-based algorithms. Furthermore, ***DyClee*** has proven to be able to cluster data sets in which the clusters exhibit different levels of density. Overlapping distributions were also tested and ***DyClee*** shows to be able to correctly cluster samples in these situations.

DyClee achieves all these properties while being computationally efficient and capable of dealing with large data sets. In fact, most of the techniques suited to cluster complex data distributions need to process the data set as a whole, which make them unsuitable for application in industrial environments or in processes in which data are generated continuously. On the contrary, ***DyClee*** is not only able to process complex distributions but it does it online taking data samples in batch or stream.

The next chapter will illustrate ***DyClee*** approach to process dynamic data samples. In order to cluster dynamic data with the structure presented in Chapters 4 to 6, and used in this chapter to cluster static data, a preprocessing stage is introduced. In this stage information involving the temporal evolution of the variables is extracted and transformed into a representation that ***DyClee*** can then cluster.

Part III

Dynamic clustering for monitoring dynamic processes

In Part II ***DyClee*** was presented and its capabilities illustrated using several data sets. These tests have shown that ***DyClee*** handles non-convex, multi-density distributions and achieves outlier rejection even in highly overlapping situations. In this part ***DyClee*** is applied to the monitoring of dynamic processes. The first chapter (Chapter 8) presents this thesis proposal for feature generation capturing dynamic characteristics. The generated features are used by ***DyClee*** to cluster the data acquired on the process and identify the operational modes. Chapter 9 shows how ***DyClee***'s clustering results can be used to automatically generate a discrete event model of the process. The last chapter presents the results of the application of the developped algorithm on industrial benchmarks.

Chapter 8

Dynamic feature generation via trend extraction

In this chapter the process of transforming a data series into a trend based representation is fully described. Until this point data samples have been analyzed as time point wise samples arriving as a whole or in a stream. Nevertheless, for an industrial application, data samples are usually part of a causal data stream from which trend information can be extracted. A trend is a time-dependent variation of a process variable which gives information about the general direction and tendency of a variable. Since such information may be very useful to supervision purposes, a trend analysis preprocessing stage is incorporated into ***DyClee***.

The preprocessing stage transforms the time series into a much richer representation that includes information about how the variables are changing in time. This quantitative representation is formed of so-called *episodes*. Filtering and trend recognition processes are involved in finding the correct episode representation for a data series.

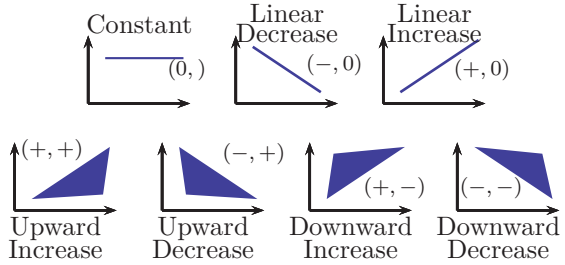
This chapter is organized as follows. First, a brief introduction about trends and their use for diagnosis and supervision purposes is presented. Second, the episodes are formally introduced giving special attention in the trend representation formalism. Then, the signal filtering stage is explained. Finally a dynamic window splitting technique is proposed to describe complex continuous signals.

8.1 Introduction

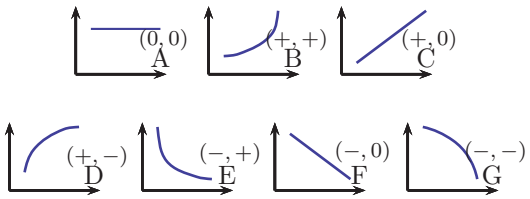
Qualitative reasoning proposes to model the physical world, characterized by continiously varying quantitative variables, in a symbolic manner [Tra and Dag 03]. Qualitative based representations are a user friendly representation that have been successfully applied in the fields of process monitoring and fault diagnosis [Tra and Mil 97, Ren *et al.* 01, Mau *et*

al. 03, Das *et al.* 04, Mau *et al.* 05, Mau *et al.* 07, GA *et al.* 09, Mau *et al.* 10, Gam *et al.* 14b]. In general Qualitative Trend Analysis (QTA) approaches are composed of three components: (i) a language for trend representation, (ii) a methodology to extract the trends and (iii) a classification methodology to map trends to process situation assessment [Mau *et al.* 10].

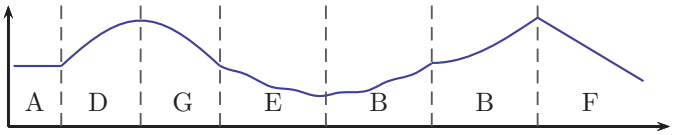
QTA is a non model-based technique that exploits historical data of a process to characterize its behavior. The use of a language reduces the complexity of system states by allowing only a finite set of qualitative descriptors [GA 12]. Those descriptors have been historicaly called *primitives* [Che and Ste 90, Mau *et al.* 07] or **episodes** [Jan and Ven 91, GA *et al.* 09, GA 12]. Figure 8.1(a) shows the triangular episode language proposed in [Che and Ste 90] and Figure 8.1(b) the seven primitives language proposed in [Jan and Ven 91]. Both representations are based on the signs of the first and second derivatives of the analyzed variable. An example of a trend represented qualitatively with the seven primitives language is shown in Figure 8.1(c).



(a) Triangular episode language



(b) Seven Primitives language



(c) A trend represented qualitatively by primitives

Figure 8.1: Qualitative representations used to represent trends

Polynomial fit-based methods have been chosen for the trend extraction task owing to its shorter computational time and higher robustness to noise. The main advantages of the qualitative representation are its interpretability, the complexity reduction and the robustness in presence of low to medium amounts of noise. On the contrary, one main drawback is the lack of differentiation of episodes following the same qualitative trend, i.e. the concept of magnitude is completely lost. See for example the two different signals depicted in blue and red in Figure 8.2. Consider that these are tank level measures. It is intuitive that these two measures carry different information (the red signal indicates a fastest increase in the tank level) and yet their qualitative representation is the same.

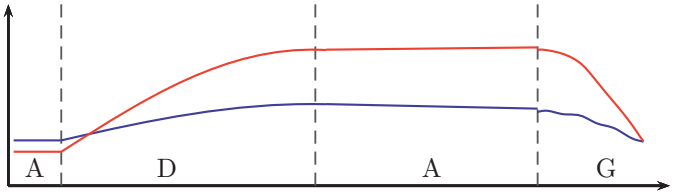


Figure 8.2: Different time series with the same QTA representation

In order to overcome this issue, this thesis proposes a quantitative approach for trend representations. As said before, most of the QTA approaches make use of polynomials to represent the trends. Specifically, they use the sign of the linear and quadratic terms to find the sign of the derivatives and hence the corresponding primitive, as proposed in [Das *et al.* 04]. However, in these works, the value of the polynomial coefficients is not used.

On the contrary, this thesis proposal is to use the polynomial coefficients as input to the classification task rather than the pure qualitative abstractions. The following sections explains this proposal.

8.2 Dynamic behavior representation

Data streams take the form of time series providing the values of the signals measured over time on a given process at each sampled time. In order to extract trend information, this work proposes to process each time series x_i into *episodes*, to generate an abstraction of the original signal into a *qualitative-like*, yet quantitative, representation.

The term episode have been largely used in the sense of a qualitative descriptor of a given representation [GA *et al.* 09]. In this thesis the same term is kept but the proposed representation is quantitative.

Episodes are defined by three elements: a trend context TC, a set of auxiliary variables AV and a time interval T_i leading to (8.1):

$$e(x_i) = \{TC, AV, T_i\} \tag{8.1}$$

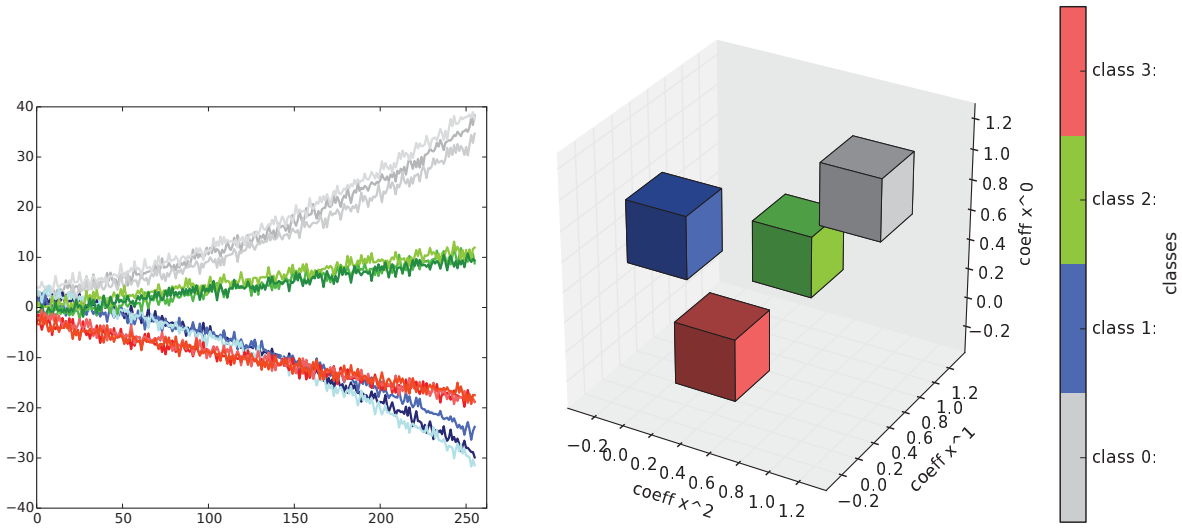
As introduced before, to find the trend context, polynomial fit can be used. Instead of using an entirely qualitative representation based on an alphabet of primitives, we use the polynomial coefficients. The use of coefficients allows us not only to retain all the qualitative information about the trend (constant, decreasing, increasing) but also to incorporate a magnitude to it. This Quantitative-Qualitative Trend Analysis is denoted by $2QTA$. The trend context TC, for a polynomial fit of order n, is the n-dimensional vector $[c_0, c_1, \ldots, c_n]$.

Auxiliary variables capture information related to the data samples covered by the episode. The importance of this complementary information in finding structural similarity between

trends was established by Angstenberger in [Ang 01]. Some possible variables to be used are average value, standard deviation, slope, curvature, smoothness, etc. Finally the time interval place the trend information in a specific time related context.

It is worth noting that the validation of a polynomial fit is done by comparing the polynomial fitting error to the time series noise variance. If the noise variance (measured as explained below in subsection 8.3) is bigger than the variance of the approximation residuals, the fit is accepted.

Example 8.1 (Clustering using trend abstraction)**.** A set of twelve synthetic signals following linear and exponential dynamics are generated. These signals have been contaminated with random white noise as seen in figure 8.3(a) where the increasing dynamics are plotted in gray (exponential) and green (linear) and the decreasing dynamics in red (linear) and blue (exponential). For the human perception finding the difference between the set of signals in the lower half of the figure (red and blue) can be difficult. Classic clustering approaches using static features are unable to separate this two groups. By using episodes, ***DyClee*** is able to clearly differentiate the two dynamics as can be seen in Figure 8.3(b), were the found μ-clusters with corresponding colors are shown.



(a) time series

(b) Episode based classification

Figure 8.3: Illustrative classification task of noisy time series using 2QTA

8.3 Signal filtering and noise variance estimation

In order to filter the input signal and find out the noise variance (used to validate the polynomial fit) an optimization process over the discrete cosine transform is performed. Consider

the smooth one dimensional signal $\hat{y}$, corrupted by zero mean Gaussian noise ϵ. The measured signal y is:

$$y = \hat{y} + \epsilon \tag{8.2}$$

[Gar 10] proved that the smoothed signal $\hat{y}$ can be expressed as:

$$\hat{y} = IDCT\left(\left(I_N + s\Lambda^2\right)^{-1} DCT(y)\right) \tag{8.3}$$

where s is a real positive scalar that controls the degree of smoothing known as *smoothing parameter*, $\Lambda = diag\left(\lambda_i \cdots \lambda_N\right)$ is the matrix whose diagonal is given by the eigenvalues $\lambda_i = -2 + 2cos\left((i-1)\,\pi/n\right)$, as defined in [Yue 05] and DCT and $IDCT$ are the n-by-n type-2 discrete cosine transform and the inverse cosine transform matrices respectively.

The variance of the noise can then be determined from the smoothed signal that minimizes the residual sum of squares $\|\hat{y} - y\|^2$ as:

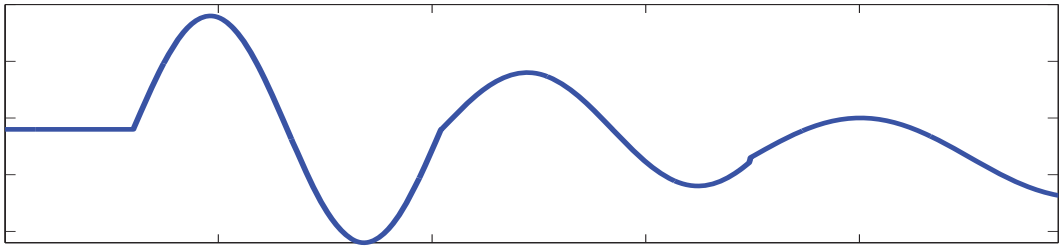
$$\sigma^2_{noise} = \|\hat{y} - y\|^2/(N-1) \tag{8.4}$$

8.4 Adaptive time window for episode representation

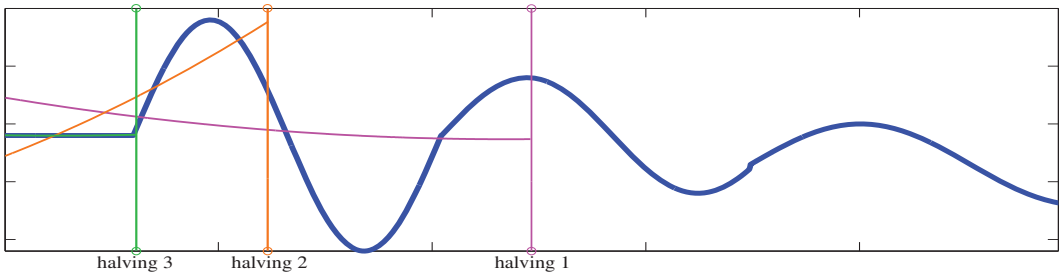
$2QTA$ based episode representation allows one to describe dynamic behaviors that may develop in different time scales, by means of polynomial coefficients. According to the Weierstrass approximation theorem [Sto 48], a continuous function defined in a closed interval, can be approximated as closely as desired by a polynomial of sufficient high order. If the process dynamics is highly variable, finding the polynomial coefficients describing the whole signal may be computationally expensive. An alternative is to split the function into smaller pieces that could be described by lower order polynomials.

The proposed algorithm makes use of polynomials of order $n \in \{0, 1, 2\}$. If the candidate polynomial does not describe properly the signal, i.e. if the variance of the error introduced by the approximation process is bigger than the variance of the signal noise, the time window is shrunk and the polynomial approximation used again to describe the behavior in the current time window.

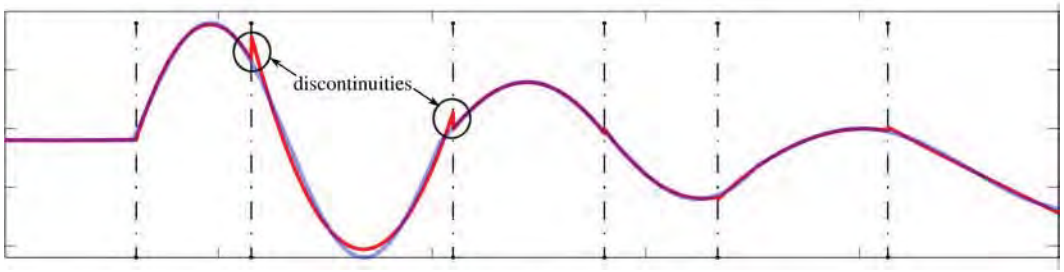
Polynomial aproximation algorithms based on window halving, also known as *interval halving*, have proven to be useful in shrinking the time window and finding an appropriate splitting point [Das *et al.* 04]. This method, however, has two important drawbacks. First, if the signal change point is located close to the start of the window, several iterations are necessary to localize it. If it is located at the end, multiple iterations and polynomials must be used to describe a portion of the signal that could be described by one polynomial only.



(a) Input signal to the IH method.



(b) To correctly describe the constant segment at the beginning of the signal, the halving procedure is applied three times.



(c) The final representation found by the interval halving method. Discontinuities are found at the beginning of the 3th and 4th segment

Figure 8.4: Interval Halving algorithm for window splitting and polynomial characterization

Second, if the change in the signal does not occur at specific times that divide the halved window, discontinuities could be introduced by the polynomial approximation. To clarify the idea, see Figure 8.4, e.g. to find the constant fragment at the start of the time series three iterations are needed. Figure 8.4 also shows the discontinuities that can be introduced in the signal as illustrated at the beginning of the third and fourth fragments of the figure. To diminish the discontinuities, weights can be used in the polynomial fit giving more importance to the fit of border samples.

To overcome these drawbacks a first approach using wavelet based multi-resolution analysis was tested and the results can be found in [Bar *et al.* 15]. The outcome is a partition of the signal favorable to polynomial characterization, nevertheless the selection of the right wavelet family for signal denoising and splitting is not trivial. This problem can be overcame eval-

uating several families and choosing the one that gives the best-fit. However, this approach involves multiple iterations which decrease the general speed of the algorithm and increase the complexity from $O(N)$ to $O(w*N)$, where w is the number of considered wavelets, restricting its applicability.

To surpass this obstacle a new approach was elaborated based on the search of those points related with behavioral changes in the measured signal. The splitting points are found by making use of peak detection in both the original and the smoothed signal. The smooth signal is found as described in section 8.3 and the exact method to found the splitting points is explained below.

The peak detection algorithm used by ***DyClee*** detects discontinuities in the signal and also local maxima, minima and inflection points. It uses the measured signal y and the difference between consecutive points of the smoothed signal $\hat{y}$:

$$d\hat{y} = |\hat{y}(t) - \hat{y}(t-1)| \tag{8.5}$$

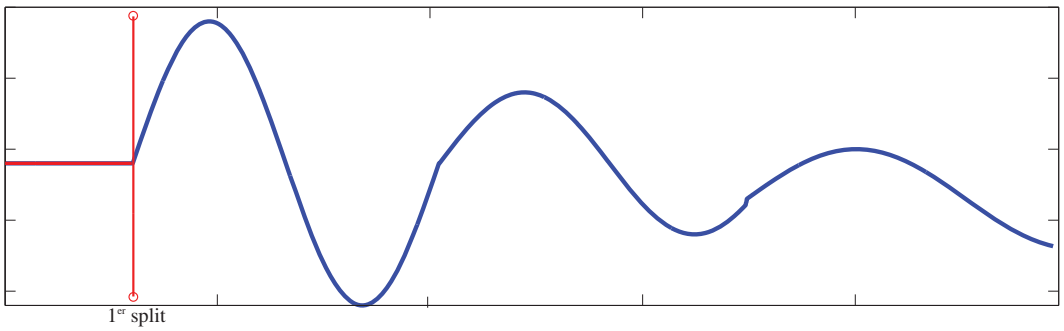
Notice that the optimization procedure described in section 8.3 provides both the noise variance estimation (equation (8.4) and the smoothed signal $\hat{y}$ (equation (8.3)).

Peaks in y correspond to local maxima and minima points and peaks in dy to inflection points. We are only interested in finding the first point of change since it establishes the time window fragment best suited for polynomial characterization. Once the point is found, the current window is sliced and the first portion is submitted to the polynomial fit. The rest of the window is updated with the data arrived from the stream before been processed again by the peak detector. In order to find the first point of change in the signal, we propose to measure the average value of the m first points in the signal and characterize the signal behavior in the following way:

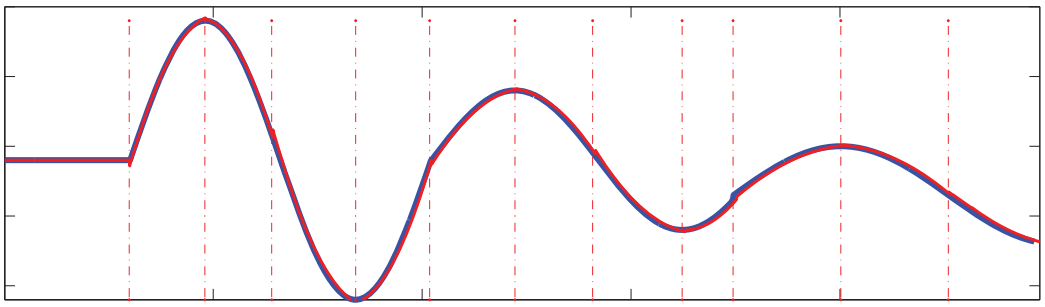
$$behavior \text{ is } \begin{cases} constant & \text{if } \|avg(\hat{y}_0, \hat{y}_1, \cdots, \hat{y}_m) - y_0\| < 2 * \sigma^2_{noise} \\ increasing & \text{if } avg(\hat{y}_0, \hat{y}_1, \cdots, \hat{y}_m) - y_0 > 2 * \sigma^2_{noise} \\ decreasing & \text{otherwise} \end{cases} \tag{8.6}$$

where $avg(\hat{y}_0, \hat{y}_1, \cdots, \hat{y}_m)$ represents the average of the first m points of $\hat{y}$ and y_0 represents the first sample of the current window. If the signal behavior is found to be increasing, the next signal maximum is chosen as splitting point, in a similar way if the behavior is decreasing the next minimum will be the splitting point. In the case of constant behavior, the point where the signal changes more than $2 * \sigma^2_{noise}$ units above or below the constant average will be selected as splitting point.

To illustrate the results, the method was applied to the signal of Figure 8.4. The results are shown in Figure 8.5. The signal fragments found are quite appropriate to low order polynomial representation. This result outcomes the representation found using the interval halving method. One advantage of the presented method is that non symmetrical graphs can be properly described, since unimodal behavior is split.



(a) To correctly describe the constant segment at the beginning of the signal, only one split is necessary with the peak detection method.



(b) The final representation found by the proposed splitting method.

Figure 8.5: Proposal of window splitting for polynomial characterization

8.5 Trend abstraction and clustering synchronization

Episode abstraction is processed for each signal or feature independently providing the so called local episodes. A proper synchronization of local episodes and their time frames must be performed to generate the input that can be applied to the clustering algorithm ***DyClee***, i.e. a $d-$dimensional vector E. Figure 8.6 shows two signals exhibiting different behavior, characterized by local episodes e_1^j and e_2^i in non synchronized time frames. In this example, the vector E that must be used as input to the clusterer is a $2d$ vector.

Episode synchronization is performed by considering behavioral changes in any dimension. The global behavior is hence described by a sequence of *epochs* for which E changes value.

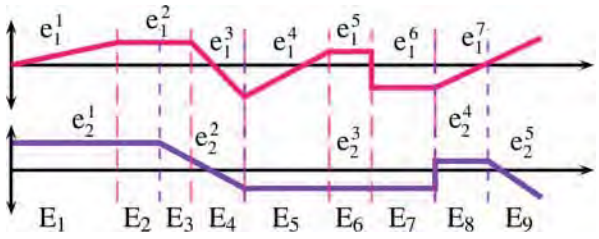


Figure 8.6: Episode synchronization

In the example of Figure 8.6, the first epoch is $E_1 = [e_1^1, e_2^1]$, then the second epoch is $E_2 = [e_1^2, e_2^1]$ and so on.

The synchronization procedure is performed on-line and generates the values of E as soon as a local episode changes. Having dynamic time frames allows episodes of any duration, even really long episodes. To avoid slow detection time through long episodes, a maximum window size `Max_ws` is established as reference for the density-based algorithm. `Max_ws` is used as upper limit to generate E and start diagnosis hence ensuring that the density-based analysis are performed within appropriate security intervals. At the same time episodes are allowed to evolve to follow process dynamics as illustrated in Example 8.2.

Example 8.2. Let us consider a system with three measured variables for which 2000 samples has been collected as depicted in Figure 8.7. This features present different dynamics and are analyzed independently by ***DyClee*** 2QTA. The trends cuts found by ***DyClee*** are shown in Figure 8.8. Each cut is depicted in color in the own signal and then in gray in the others for illustration purposes. It is worth remembering that each episode change in one of the signals, implies a new Epoch vector generated and send to the density-based clustering stage. Note also that in $n = 1700$ a security Epoch is created, this Epoch is generated since no changes have been detected in the green signal in a time window of `Max_ws`. This Security epoch guaranties that ***DyClee*** always diagnose the system with a frequency less or equal to 1/`Max_ws`.

8.6 Summary

In this chapter a proposal for feature generation via trend extraction was presented. This method allows us to capture the information about the evolution of the monitored characteristics into episodes by making use of polynomial representations. The episode abstraction is done for each monitored characteristic and then packed into epochs which are analyzed by the clustering algorithm presented in Part II.

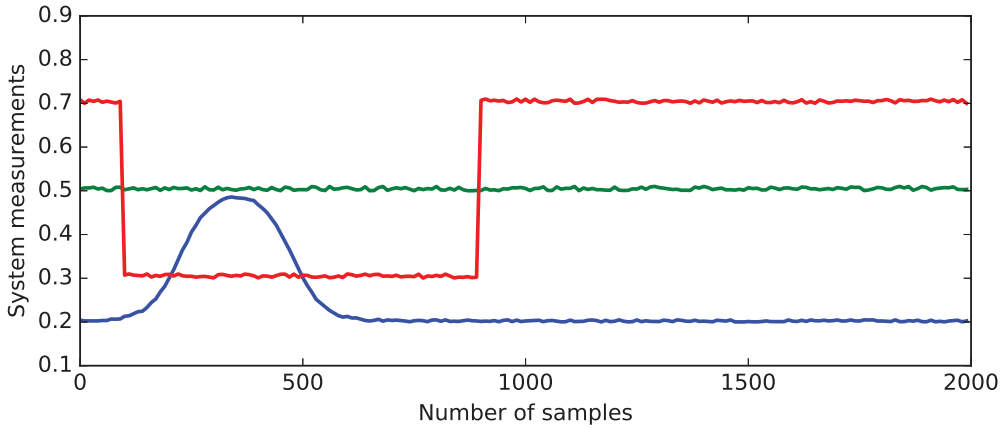


Figure 8.7: System measurements as collected

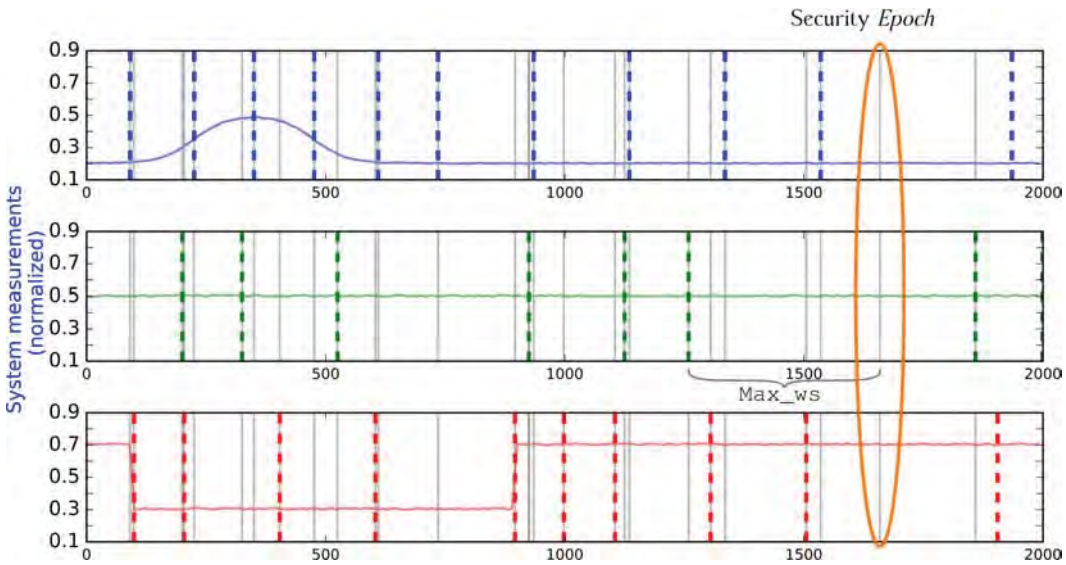


Figure 8.8: Example of ***DyClee*** synchronization over the system features

Chapter 9

Learning an adaptive discrete event model of the process from clustering results

In Part II a clustering based method for tracking evolving systems was presented. The developed algorithm, ***DyClee*** is able to consider system measurements as static features (discarding the temporal related information) or as dynamic features by the use of the episode representation arising from $2QTA$ (See Chapter 8) in order to characterize the system behavior. This chapter presents a proposal for using a discrete event model coupled with the continuous dynamic clustering in order to improve the supervision capabilities. Interestingly, such model can be learned from the clustering produced by ***DyClee***.

As deeply explained in the previous chapters, classification techniques establish a representation of the process behaviors by grouping samples in the feature space. As was introduced in chapter 3 these features are constructed in such a way that they make samples distinguishable with regard to a certain concept (or set of concepts). In the case of clustering such concepts are not known in advance, so the feature space have not been selected in advance to accomplish an specific goal. Clustering methods usually start from the measure space and find relations that allow an a posteriori selection of features. These methods are specially suited for complex processes where physical models are complex to develop. Clusterers characterize the system behaviors as a cluster partition and then assess the process situation online by tracking the process measurements [Kem *et al.* 06]. The concepts of 'normal', 'degraded', 'failure' are unknown to the clusterer but known to the human operator. Clustering techniques for supervision must be able to support an interpretable macroscopic view of the process focused on help the operator in the decision making stage by presenting the information about the current process state and the possible future states.

The inclusion of a discrete event model follows two objectives: improving clustering results interpretability for decision-making purposes and improving the fault detection capabilities by the inclusion of event related dynamics.

As introduced in Chapter 2, a diagnostic system generates indicators about faulty behavior in the form of symptoms that should be interpreted in order to find the correct decision about the actions to take over on the system. In non-supervised approaches this interpretation comes from a human expert who analyses the cluster information and assigns an appropriate label to it. The visualization of the clustering results in a simple, yet complete representation becomes then necessary.

DyClee is able to describe and track the current behavior of the system but it has no information regarding the correspondence of the 'observed behavior' it finds with the desired behavior. New detected clusters might or not represent faulty states, so the fault diagnosis cannot be performed without the expert interpretation.

The developments made in this chapter are based on the following hypotheses:

Hypothesis H9.1. The system expert is able to provide the labels of the (finite) set of nominal operational modes as well as the set-points changes to transition between them (*control events*).

Hypothesis H9.2. Changes in the system set-points happen in a much slower timescale than system dynamics, i.e. the system reaches and remains in a steady state between two set-point changes.

Hypothesis H9.3. The tuple formed by the observed operational mode (continuous state) and the control reference (discrete state) of the system is considered to be unique.

In complex systems faults can be detected when the process enters in a failure mode. Using ***DyClee*** clustering faulty states can be observed and characterized, nevertheless since it does not consider the system references as input, faults due to unexpected transitions between different operational points or violations to state sequences cannot be detected. Under hypothesis H9.1 and H9.2 these faults can be detected thanks to the DES model. In specific, three types of event-related faults are detected: 1) occurrence of an control event not followed by the actual transition, 2) state transition without the occurrence of the control event, 3) state transition that does not correspond to the applied control event.

Example 9.1. To illustrate the three kind of faults the blue car example (3.3) will be used. Let us assume that the change from the stopped state to the moving forward state is driven by the control event "accelerate" and the opposite transition by the event "slow down" as shown in Figure 9.1.

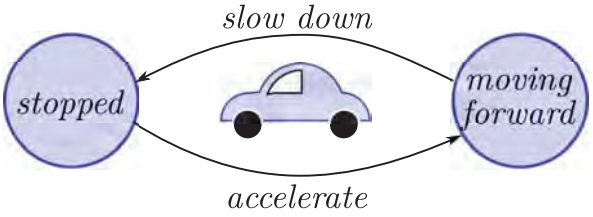


Figure 9.1: Blue car event driven transition

- If the car is in the *stopped* state and the "accelerate" event is detected but no state change is evidenced, a fault of type 1 has occurred.
- If the car is in the *stopped* state and it starts moving without any control event, a fault of type 2 has occurred.
- If the car is in the *moving forward* state and after an "slow down" event the car starts moving backwards instead of stopping, a fault of type 3 has occurred.

Fault detection is possible under hypothesis H9.3. This hypothesis implies that, in the case where for a operational mode two different control reference (events) are observed a new label is created to differentiate this behavior.

In this chapter a methodology to automatically generate a discrete event model of the monitored process from ***DyClee*** clustering results is presented. This model represents system behaviors using a timed automaton in which the states represent ***DyClee*** natural clusters and the transitions the states reacheability. The model is updated each t_{global} time units, as a new clustering is provided by the density-based stage. Qualitative information about the system states is included in a graphic compact representation that can help, if necessary, in the process of decision making. This chapter is organized as follows: Section 9.1 introduces the discrete event systems (DES) and presents the chosen DES model formally and with practical examples. Section 9.2 explains the construction process of a DES from ***DyClee*** clustering results. Finally section 9.3 summarizes the chapter.

9.1 Discrete event system models for process diagnosis

Approaches for diagnosing DES have been proposed in both the AI and control engineering literature and covering systems that are discrete by nature as message (alarm) based systems as well as continuous systems (after quantisation). Diagnostic algorithms model the process states as normal and abnormal using whether physical principles, expert knowledge (rule-based) or data mining mechanisms based on labeled samples. For the purposes of diagnosis, large scale dynamic systems described by continuous variables can often be viewd as DES at

some level of abstraction [Sam *et al.* 96]. The set of the process possible situations (and the related concepts) form a finite set, this fact makes DES especially suited for represent them [Wai *et al.* 00]. DES consider the state space as a discrete set in which the state transitions are *event-driven* [Cas and Laf 10].

Application of DESs for supervisory tasks are varied. A fuzzy automaton was presented in [Kem *et al.* 06] in which the system trajectory is described by a sequence of classes to which the actual situation belongs. This automaton was used to represent a reference behavioral model of the process and detect process deviations at state level (unknown behavior) and at transition level (unexpected transitions). Sampath *et al.* achieve system diagnosis by first building a set of finite-state models of the subsystems to be diagnosed and then building a finite-state machine that will perform as *diagnoser* of the hole system based on indicator events [Sam *et al.* 96]. A decentralised approach for on-line diagnosis was also proposed in [Pen and Cor 05] to reduce diagnosis computational requirements. In [Pen and Cor 05], component diagnoses are progressively merged to obtain subsystem diagnoses and the final system diagnosis. Yet another decentralized approach was proposed in [Cor *et al.* 07] in which global diagnosis is represented by the set of diagnoses of its transition-independent subsystems.

Gaudel *et al.* use a hybrid particle petri net to diagnose system modes (normal and abnormal behaviors), represented as discrete states with continuous dynamics, in which faults are considered as unobservable events [Gau *et al.* 15]. In [Sub *et al.* 14], the authors deal with the issue of extracting temporal patterns (in the form of chronicles) that are common to a system behavior or situation. The extracted *chronicles* describe each situation which allows its further use for situation assessment purposes. Specifically, fault diagnosis can be accomplish by analyzing the system flow of observations in comparison with the already learned chronicles.

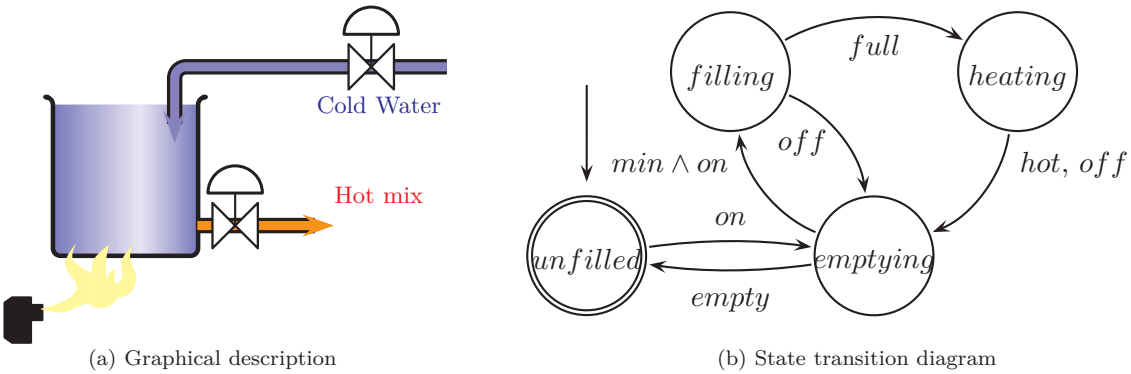
DyClee allows to represent temporal related information inside the clusters and consequently the chosen DES should be able to incorporate temporal information associated to the transitions between the characterized behavioral modes. Complex process behaviors are usually submitted to timed-based or event-based constraints. Conditions as *'the product must heat up to 100 degrees'* or *'the mix must cold down 20 minutes before its injection into the mold'* can not be modeled by using basic DESs. Among the timed DESs, approaches based on timed automata (stochastic or deterministic), timed petri nets and $(max, +)$ algebra methods can be found [Lun and Sup 02, Sup *et al.* 06, Cas and Laf 10]. To merge time-driven dynamics with event-driven dynamics timed automata (with guards) are chosen as DES formalism in the following.

9.1.1 Timed Automata

In order to introduce a timed automaton with timed guards the definition of automata and its extension to use timed transitions is first introduced. An automaton $\mathcal{A}$ is represented by the tuple $\langle E, \mathcal{S}, s_0, Tf, \mathcal{S}_m \}$ where S is a finite set of states, E is a finite set of events, $s_0 \subseteq S$ is the initial state, $Tf : \mathcal{S} \times E \to \mathcal{S}$ is the *transition function*. The automaton starts in an initial state and then if $\langle s, a \} \in Tf$ the automaton changes state from s to s'. $\mathcal{S}_m$ is the set of accepted or final states called *marked states*. To clarify this definition an example is given below.

Example 9.2 (A water heater automaton)**.** Consider a water heater as the one shown in Figure 9.2(a). This system is formed by a tank, a burner and two pipes. The system behavior is the following: the tank is filled with water until it is *full*, then the burner is activated until the water gets *hot* and then the tank is drained until it reaches a *minimum* level. In addition if the process is turned off the tank should be completely drained.

The state transition diagram describing the process behavior is shown in Figure 9.2(b) where nodes represent states and labeled arcs represent transitions between these states. This graph is a graphical representation of an automaton. The set of nodes is the state set $\mathcal{S} = \{$ '$unfilled$', '$filling$', '$emptying$', '$heating$' $\}$, the labels of the arcs are elements of the event set $E = \{$'$empty$', '$full$', '$minimum$', 'hot', 'off', 'on'$\}$ and the arcs provide a graphical representation of Tf. $\Gamma(s)$ is the set of all events e for which $Tf(s, e)$ is defined. The marked state is '$unfilled$'.



(a) Graphical description

(b) State transition diagram

Figure 9.2: Water heater

Timed automata formalism is an extension of automata in which transitions from any state in S to any other state in S could have temporal constraints. Adjoining time-driven dynamics allows to represent information such as how long the system stayed in a given state or what is the time interval between two events.

Timed constraints can be added to the transitions of the automata using a finite set of clocks. Before giving the formal definition, the concept is illustrated using the example 9.2.

Example 9.3. The condition '*the mix should boil for at least 20 minutes*' can be included in the automaton using a clock c set to zero at the instant in which the mix reaches the boil temperature ('*hot*' event) and allowing the transition from *'heating'* to *'emptying'* only if c is bigger than twenty. The state diagram in Figure 9.3 describes the process including this new condition.

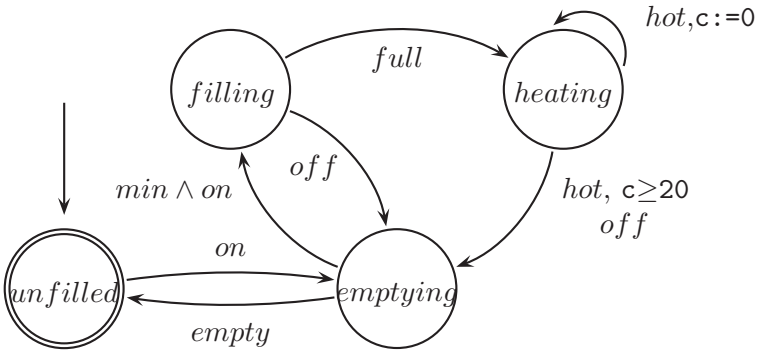


Figure 9.3: Water heater state transition diagram with a time constraint

The formal definition of timed automaton is given below.

Definition D9.1 (Timed Automaton [Alu and Dil 94])**.** A time automaton $\mathcal{A}_t$ is a tuple $\langle E, \mathcal{S}, \mathcal{S}_0, Tf_t, \mathcal{S}_m, \mathcal{CK}\}$ where S is a finite set of states, E is a finite set of events, $\mathcal{S}_0 \subseteq S$ is a finite set of start states, $Tf_t : \mathcal{S} \times E \times 2^{\mathcal{CK}} \times \Upsilon(\mathcal{CK}) \rightarrow \mathcal{S}$ is the *timed transition function.* $\mathcal{S}_m$ is the set of accepted or final states and $\mathcal{CK}$ is a finite set of clocks.

When the automaton is in the state s and the event a is detected, the automaton changes state from s to s' iff the clock constraint δ is fulfilled, that is the tuple $\langle s, a, \iota, \delta\} \in Tf_t$. The subset $\iota \subseteq \mathcal{CK}$ makes the clocks to be reset with this transition.

Example 9.4. For the example 9.3 the timed transitions are:

$$\begin{aligned} Tf_t\,(heating, hot, \{c\}, \{\}) &= heating \\ Tf_t\,(heating, hot, \{\}, \{\mathsf{c} \geq 20\}) &= emptying \end{aligned}$$

Example 9.5. For the example 9.3 the full transition table is depicted in Figure 9.4. In this case the timed constraints are given by design and specified next to the transition triggering event.

departure \ arrival	$unfilled$	$emptying$	$filling$	$heating$
$unfilled$		on		
$emptying$	$empty$		$min \wedge on\ 5$	
$filling$		off		$full$
$heating$		$hot\ [c \geq 20]$ off		$hot\ [c := 0]$

Figure 9.4: Transition table for example 9.3

9.2 Modeling the process behavior from clustering results

In the previous chapter a method for describing time-varying features using episodes was presented. ***DyClee***'s μ-clusters keep temporal information that is reflected in the final cluster profile. The DES is built from the clusterer transition matrix found directly from ***DyClee*** clustering results in the following way:

E: The departures and arrivals from and to natural clusters are used as events, i.e. for each characterized cluster two events are created the arrival event $Clus_i$ and the departure event $\overline{Clus_i}$. The set of events grows up incrementally as new clusters are created.

$\mathcal{S}$: The set of states of the automaton is given by the set of natural clusters. This set grows up incrementally as new clusters are created. The notation Cl_i is adopted to better distinguish the states from the cluster transitions ($Clus_i$).

$\mathcal{S}_0$: The initial state is the first characterized cluster in the case where no previous structure or knowledge of the process is available. Otherwise, the set of initial states is the set of operational points of the process.

Tf_t: The transition function is constructed incrementally as new clusters are detected. The time constraints are described as time intervals $[\tau_1, \tau_2]$ describing the minimum and maximum time of transition between two clusters (remember that the system may be cluster as being in $\mathbb{O}\mu$C during some time before entering an actual cluster). This interval is updated as more transitions are observed.

$\mathcal{S}_m$: If some knowledge of the process is available, the set of accepted states corresponds to the known operational modes. In the case where no such knowledge is available, the DES implementation try to estimate the possible accepted states, assuming that they are a subset of the normal operational.

$\mathcal{CK}$: One clock is created for each characterized cluster. Clock notation is cc_i, with i been the number assigned to the cluster.

9.2.1 Adding information to the Timed Automata

DyClee clustering results include temporal information about the cluster that can be included in the Timed Automata formalism. By default all natural clusters are considered as both possible departure and arrival states. The chosen timed constraints represent the minimum and maximum time between a cluster departure and the arrival to the next cluster. This time interval corresponds to the time in which ***DyClee*** tracks the current state as transitional or unknown, i.e. the system measurements are considered as low density samples and assigned to $\mathbb{O}\mu$C.

The transition table columns col_i and rows r_i corresponds to the system states and each cell ij stores a transition interval $[t_1, t_2]$ characterizing the time constraints $cc_i \geq t_1$ and $cc_i \leq t_2$ (with $t_2 \geq t_1$). To reset each clock a set of complementary states $\mathcal{S}^c$ is added to $\mathcal{S}$, $\mathcal{S}^T = \mathcal{S} \cup \mathcal{S}^c$. These complementary states represent the fact of *not been in* a cluster. The clock cc_i is set to zero (denoted as $cc_i := 0$) when the $\overline{Clus_i}$ event is detected and the system goes from state Cl_i to state $\sim i$, with $\sim i \in \mathcal{S}^c$. For simplicity and without loss of generality, from this point onward the superscript ^{T} is removed.

Another temporal information that is important in process monitoring refers to how long the system remained in a particular state. To better describe this information the median of the past stays (of the system in a specific cluster) is added to the Timed Automaton. The set of the cluster's median time of stay is called $\mathcal{D}$.

Reflecting clusterer evolution in the DES is crucial to keep the system automaton updated. The following explains how parametric and structural clusterer changes are represented in the DES.

Modeling structural changes

To handle the structural changes in the clusterer the following procedures are used:

Cluster creation When the k^{th} cluster is created the following changes are performed in the DES automatically:

- the state Cl_k and the state $\sim k$ are added to $\mathcal{S}$
- the counter cc_k is added to $\mathcal{CK}$
- a k^{th} row is added to the transition matrix
- a k^{th} column is added to the transition matrix
- $Clus_k$ and $\overline{Clus_k}$ are added to the event set E

Cluster elimination When a cluster is eliminated the state remains in the DES but its graphical representation changes to a smaller gray circle, which intuitively places the cluster as old.

Cluster merge When two or more clusters are merged the transition matrix is rebuilt using the union of both states. In the case that both states have arriving transitions departing from the same state the new state time constraints are set as the interval union of those of the previous transitions. The same principle applies to transitions departing from the old states to a same

destination. As explained in chapter 4 to assign a label for the cluster formed by the merging process, the labels of the merged clusters are analyzed and the oldest one is chosen as the newly formed cluster label.

Cluster split When a cluster is split into two o more clusters, the transitions are recalculated from available history (recent data). The transitions that cannot be confirmed by looking at available history will be depicted as dotted lines until be confirmed or deprecated. A transition is deprecated if in a time-span corresponding to tree times the sum of all elements in $\mathcal{D}$. Deprecated transitions do not cause loss of information under hypothesis H9.4.

Hypothesis H9.4. A system is considered as a regular system, i.e. the system behavior is repeatable. Any possible transition from a system state to another repeats with a certain frequency.

Modeling parametric changes

Changes in the cluster parameters are not reflected directly in the DES, nevertheless, since they are also important to the system operator, a descriptive table is also available. This descriptive table summarizes the following cluster information:

- Time of last assignation of system data to any μ-clusters in cluster k. This indicates the last time in which the cluster was active.
- Number of μ-clusters conforming the cluster.
- Median of cluster densities.
- Cluster center of gravity calculated as $C_g^j = \frac{\sum_k C_k^j * D_k}{\sum_k D_k}$ for each of the j features.
- Minimum of the cluster feature range
- Maximum of the cluster feature range

Cluster drift can be easily detected by the change in the cluster center of gravity C_g and/or an augmented number of μ-cluster.

9.2.2 Building the timed automaton

At each run of the density-based clustering stage a file containing the clusters information and the labels found for the analyzed time window is created. An example of the information transferred to the DES generation stage from the density-based clustering is presented in tables 9.1 and 9.2. With the information in Table 9.2 the transition table is constructed. Using this table the timed automaton is constructed following the procedure specified in subsection 9.2.1 according to the type of observed change. Then the automaton is enriched with the information in Table 9.1 in order to generate the final automaton that is presented to the user. This process will be illustrated in detail for two different test scenarios in the next chapter.

cluster	$\mathcal{D}$	active μC	Center of gravity	Range min	Range max	Merged with
1	4738	3	[0.0, 0.0, 0.84, 0.52, 0.53]	[0.0, 0.0, 0.69, 0.31, 0.25]	[0.15, 0.15, 1.0, 0.76, 0.86]	5
2	4738	4	[0.0, 0.96, 0.78, 0.54, 0.61]	[0.0, 0.81, 0.55, 0.32, 0.40]	[0.15, 1.0, 0.96, 0.79, 0.79]	
3	3100	12	[0.99, 0.95, 0.80, 0.43, 0.60]	[0.81, 0.76, 0.63, 0.22, 0.49]	[1.0, 1.0, 0.97, 0.65, 0.75]	
4	4500	7	[1.0, 0.0, 0.81, 0.38, 0.37]	[0.85, 0.0, 0.65, 0.21, 0.23]	[1.0, 0.14, 0.97, 0.56, 0.52]	
6	4738	1	[0.0, 0.0, 0.82, 0.62, 0.04]	[0.0, 0.0, 0.67, 0.47, 0.0]	[0.15, 0.13, 0.95, 0.77, 0.19]	

Table 9.1: Example of cluster metrics generated from the density-based stage

Time	7500	10000	12500	$\cdots$	442500	445000	447500	$\cdots$	592500	595000	597500	$\cdots$	957500	960000	962500
$id_{\mu C_z}$	1	1	1	$\cdots$	5	5	5	$\cdots$	8	8	9	$\cdots$	11	11	11
lab_z	1	1	1	$\cdots$	3	3	3	$\cdots$	5	5	0	$\cdots$	6	6	6

Table 9.2: Example of μ-cluster history used as input by the DES generator

9.3 Summary

In this chapter the need of a discrete event model for supervision purposes was introduced. The timed automaton formalism is introduced in section 9.1.1 and its use, for modeling process behaviors based on ***DyClee*** clustering results, is presented in Section 9.2. This adaptive model, automatically generated from clustering results, provides a high level abstraction view of the system with information about the system past and current states along with the possible transitions identified by the time of transition, the estimated probability of occurrence and the time constraints between states.

The next chapter shows that a timed automaton can be correctly generated from clustering results in a real processes benchmark and that the DES model improves clustering results interpretability. Furthermore the DES model gives the operator the information about past states as well as possible future states.

Chapter 10

Application to industrial Benchmarks

The main complexity of industrial engineering processes is the interaction of several physical domains (mechanical, thermical, hydraulics, etc.) which makes difficult their model based monitoring. In some of these processes, strong non-linearities are usually present (e.g. thermodynamic processes), which added to non-stationary parameters (e.g. pressure variation inside a boiler causes water and steam densities to change) make the processes prone to constant evolution (*drift*) [Dje *et al.* 09]. In this situation, data-based monitoring approaches are adequate.

In this chapter ***DyClee*** clustering and modeling capabilities are tested using two industrial benchmarks: a steam generator process [Oul 14] and a continuous stirred tank heater [Tho *et al.* 08]. The first section introduces the steam generator and ***DyClee*** monitoring function over its different operational modes. The second section introduces the continuous stirred tank heater and presents the modification made to the model in order to implement operational drift as well as permanent, transient and intermittent faults. Then ***DyClee*** monitoring is applied to different operational scenarios. Finally section 10.3 summarizes the main topics of this chapter.

10.1 Steam generator Process

In this section ***DyClee*** performance was validated in the steam generator process. The steam generator process is a pilot process, describing a thermal power plant on a reduced scale found at the Lille 1 University - Science and Technology under charge of the CRIStAL laboratory (previously known as LAGIS laboratory) that has already been used to validate model-based and data-based supervision approaches [Kem *et al.* 06, Dje *et al.* 09, Bot *et al.* 13].

The thermal power plant is composed of four subsystems: a tank with the water supply system, a boiler heated by a thermal resistor of $55kW$ and total volume of $170l$, a steam flow system and a complex condenser coupled with a heat exchanger (see Figure 10.1). In this thesis, only the boiler (including the water supply) is studied.

The boiler subsystem description and data were provided by the authors of [Kem 04]. The feed

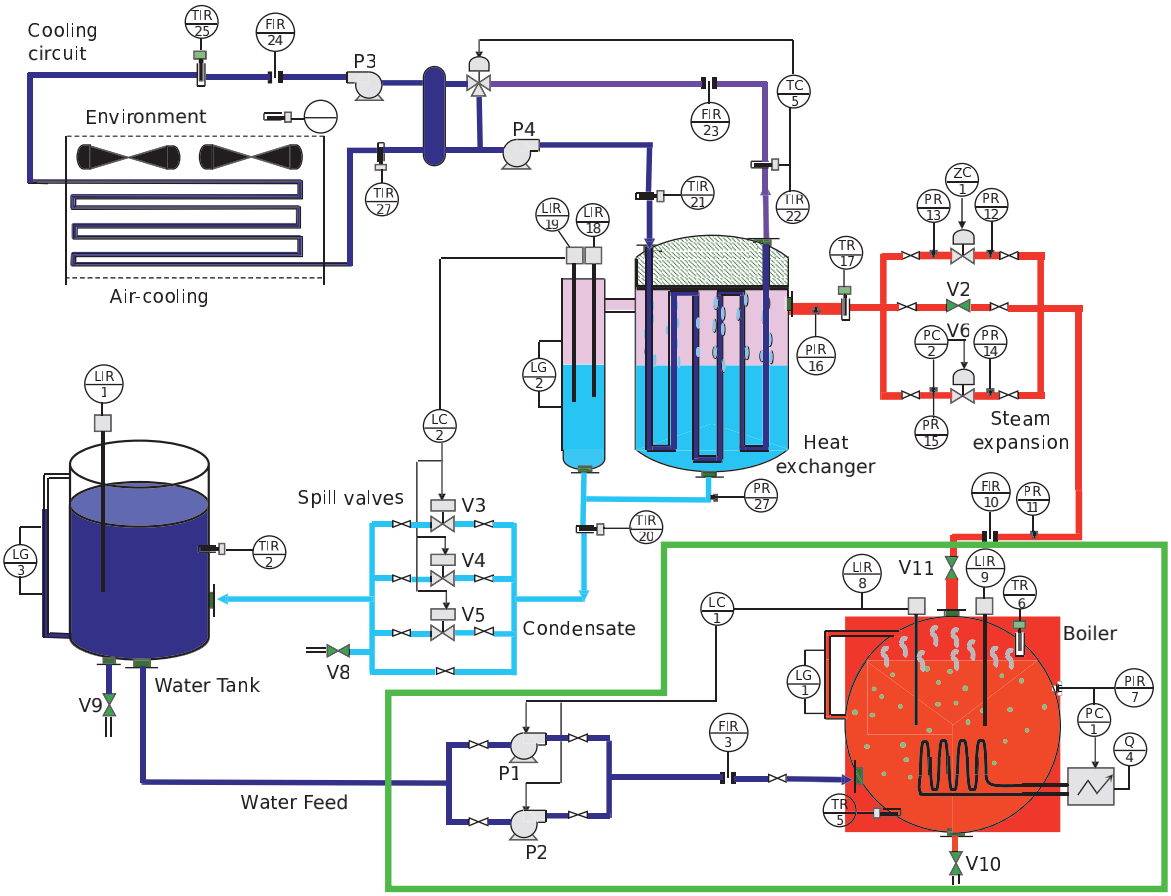


Figure 10.1: Instrumentation diagram of the thermal power plant at Lille 1 University. Modified from [Oul 14]. The green area is the subsystem under study.

water flow (sensor in FIR 3) is pumped to the boiler via a pump. In normal operation an on-off controller activates the pump P1 to maintain the water level in the boiler (sensor in LIR 8) at $\pm 3l$ of the set-point. The heat power value (Q 4) depends on the accumulator pressure (sensor in PIR 7). When this pressure drops below a minimum value, the heat resistance is activated at maximum power until the maximum pressure is achieved and then deactivated in order to keep the pressure within ± 0.2 bar of the set-point. The generated steam flow is measured with sensor in FIR 10. A description of the system outputs used for monitoring the boiler subsystem are presented in Table 10.1.

Symbol	Description	Unit	Range	Operational Point
$F3$	Supply water flow	l/s	$0-1600$	950
$Q4$	Heating power	Kw	$0-60$	60
$P7$	Boiler Pressure	Bar (abs)	$0-16$	8
$L8$	Boiler Level	l	$143-156$	146
$F10$	Steam output Flow	Kg/h	$0-100$	83

Table 10.1: Characteristiques of the steam generator outputs used for monitoring the boiler subsystem. Taken from [Kem 04].

The output variables used as input to ***DyClee*** were the feed water flow, the heat power, the boiler pressure, the boiler level and the steam flow, which were characterized as the most relevant by the process expert. Normalization were performed to homogenize the influence of each of these features in the clustering.

10.1.1 Monitoring system operation

Under normal conditions the process goes through four operational modes in order to achieve continuously steam generation. These modes are:

No regulation: Since the boiler pressure and level are inside the operational range, no regulation is needed.

Pressure regulation: When the boiler pressure drops under the low threshold, the heat resistance is activated.

Level regulation: When the boiler level drops under the low threshold, the pump P1 is activated.

Level and pressure regulation: If the water inflow has a very low temperature, the boiler pressure drops and the heat resistance is activated before P1 closure.

Data measurements of the steam generator system registred in three available data sets are depicted in Figures 10.2, 10.3 and 10.4. These data sets were used by [Kem 04] and [Bot *et al.* 13] to train, validate and test their approaches. Since ***DyClee*** is a pure unsupervised approach that performs continuously learning and recognition tasks, there is no need of a training stage. As consequence, the three data sets corresponding to training, validating and testing data were merged into a unique data set (see Figure 10.5) in which a sample time of 1 second was taken. These data were feed to ***DyClee*** for analysis. Nevertheless, for the sake of clarity, the results are presented in an incremental fashion, comparing ***DyClee*** results with the results of the other approaches. The parameters used for ***DyClee*** in this test are shown in table 10.2.

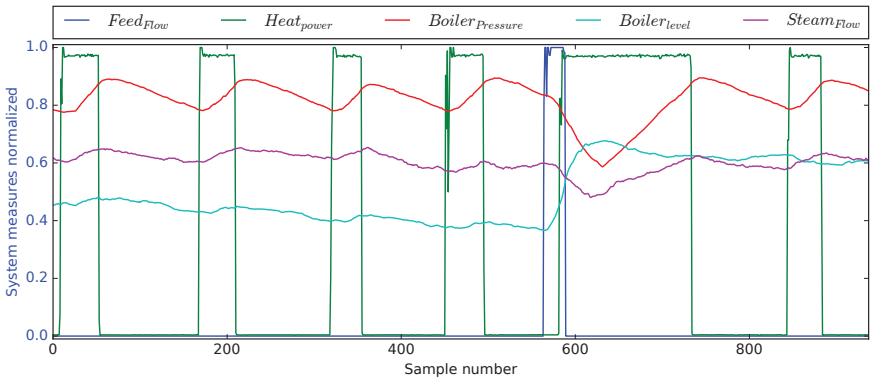


Figure 10.2: First data set: Outputs of the steam generator in normal operation mode.

DyClee clustering results for the first 1000 samples are shown in Figure 10.6 along with the system measurements. Figure 10.6 shows that ***DyClee*** finds five natural clusters in this data set. Clusters 1 and 5 correspond to the **no regulation** operational mode. Data grouped as cluster 2 corresponds to

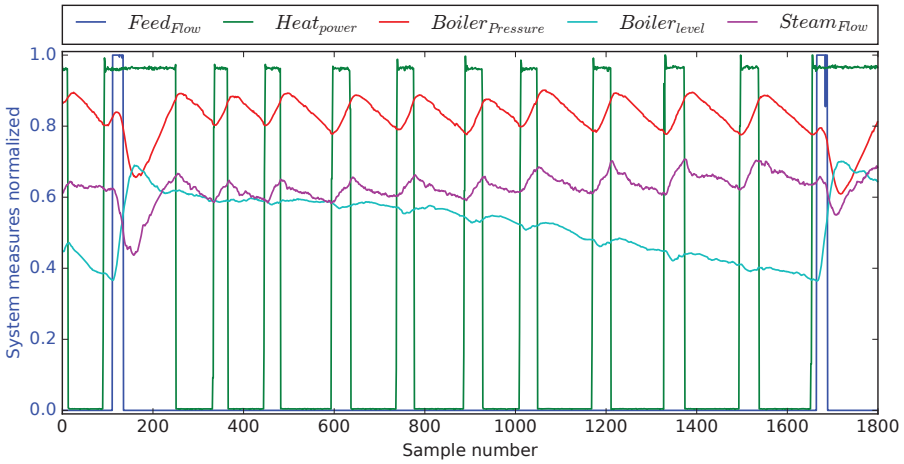


Figure 10.3: Second data set: Outputs of the steam generator in normal operation mode

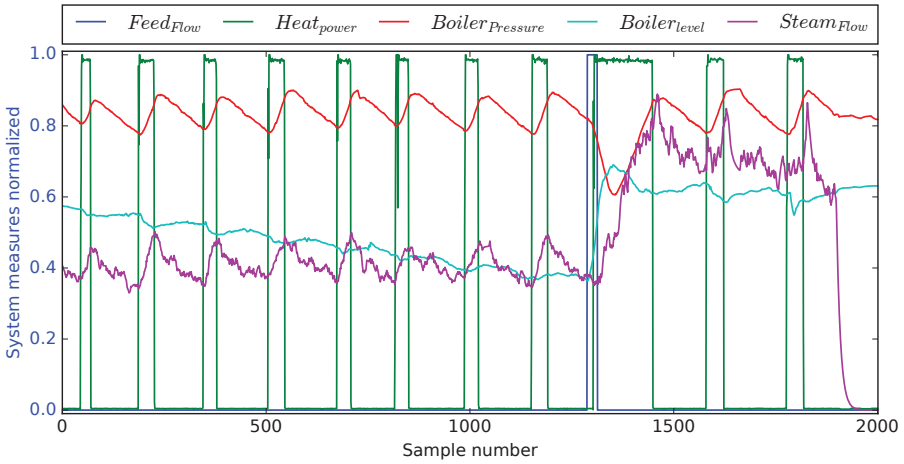


Figure 10.4: Third data set: A fault in the steam flow output is introduced at the end

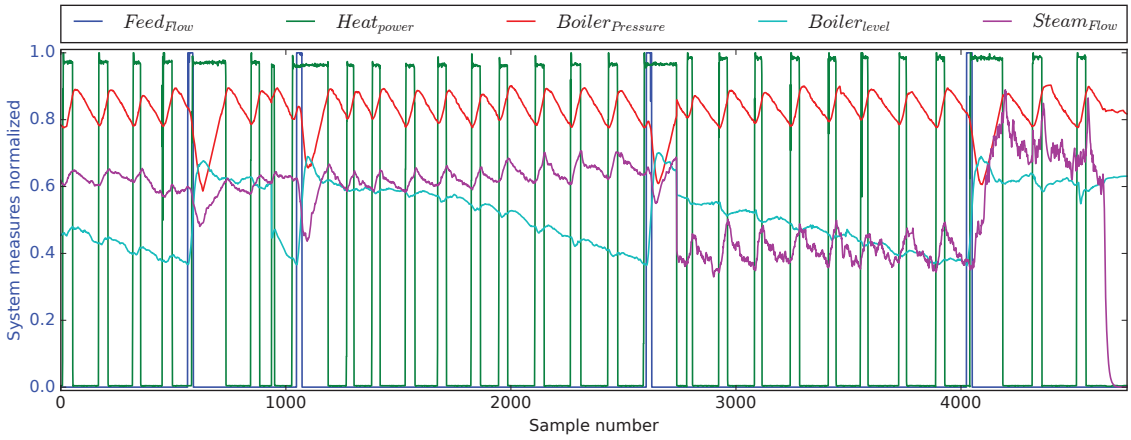


Figure 10.5: Steam generator data as presented to ***DyClee***

pressure regulation mode. Cluster 4 represents data in the **level regulation** mode and cluster 3 the **level and pressure regulation** mode.

Parameter	Value
`portion`	0.3
t_{global}	100
`multi-density`	`True`

Table 10.2: ***DyClee*** parameters used for monitoring of the steam generator process

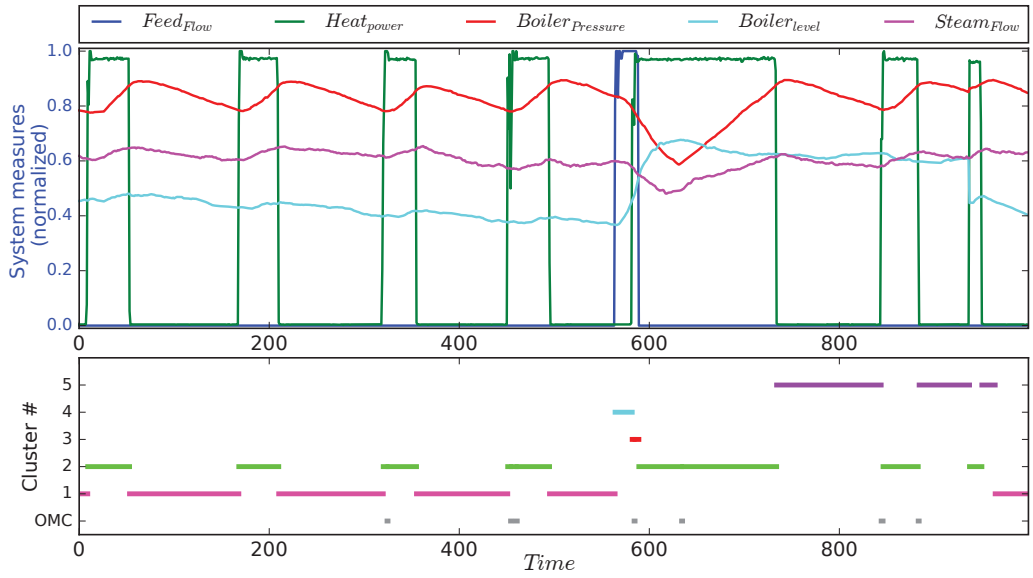


Figure 10.6: ***DyClee*** clustering results of steam generator data in normal operation mode (first 999 samples)

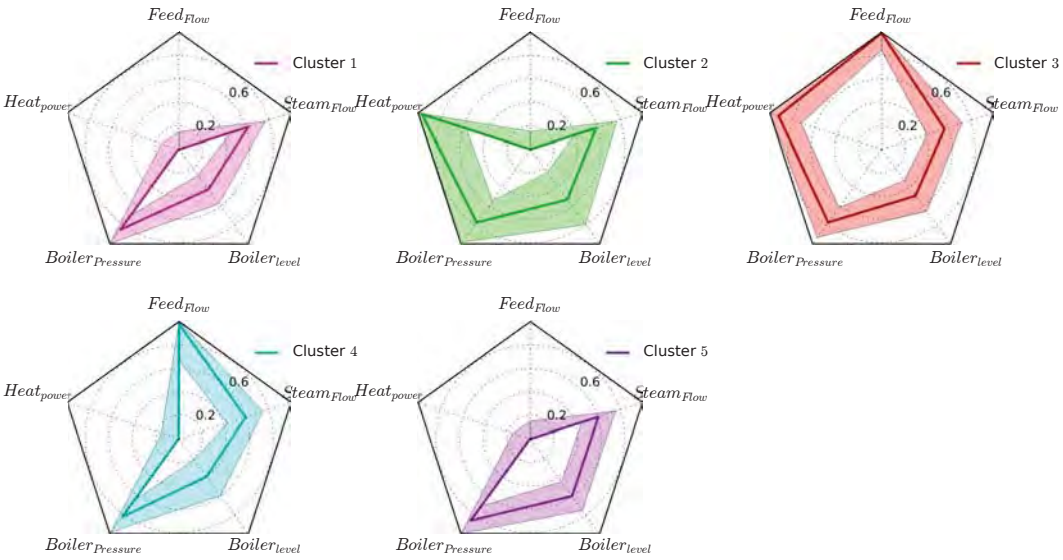


Figure 10.7: Radar representation of the natural clusters found by ***DyClee***. The thick line represents the cluster center of gravity while the shadow area represents the whole range of the cluster.

Differences between the two clusters representing the no regulation mode is caused by the difference between the measures of the boiler level. Radar plots representation has been chosen for showing the profile of the found clusters. In this plot the center of gravity of the cluster is depicted as a thick line

and the cluster's feature range as a filled area. For the first 999 samples of the data set clusters' profiles are shown in Figure 10.7. In this figure the similarity between Clusters 1 and 5 can be evidenced and also the fact that the main difference is the boiler level measurement.

In order to show ***DyClee*** μ-clusters evolution four snapshots of the μ-cluster's distribution are shown in Figure 10.8. The μ-clusters are represented as cubes centered in the μ-cluster center. For the x, y and z axes, the three features with maximal variance are chosen in order to ease graphical interpretation of the results ($Feed_{Flow}$, $Heat_{power}$, $Boiler_{level}$). The cube color represents the μ-cluster class and the μ-cluster density is represented as the cube opacity.

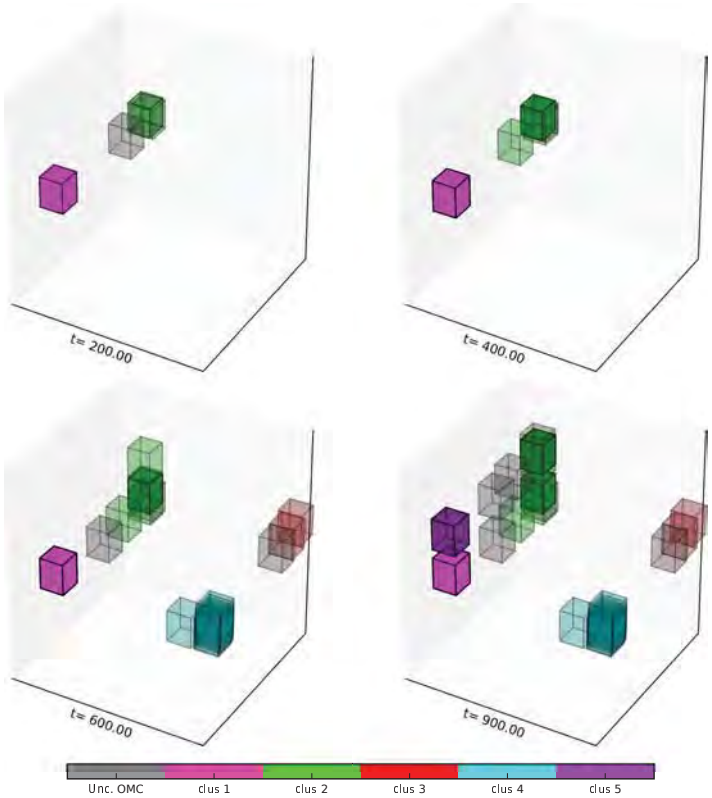


Figure 10.8: ***DyClee*** μ-cluster clustering results of steam generator data in normal operation mode (samples $0 - 999$)

It is interesting to see that clusters 1 and 5 are conformed by only one μ-cluster while Cluster 4 is formed by 8 μ-clusters. These phenomenon is explained by the dynamism described by the natural clusters. Cluster 4 represents a class in which the measured outputs change quickly. On the contrary, clusters 5 and 1 represent the system in much slower dynamic. These clusters, as well as Cluster 2, exhibit a drifting behavior, common in thermodynamic systems.

Figure 10.9 shows clustering results up to $t = 2700$, that is, at the end of the second data set. Due to their drifting behavior clusters 1 and 5 have merged. This merge is directly reflected in the clustering results graph in which the label 5 has been replaced by label 1 by the clusterer update process. The cluster's profiles are shown in figure 10.10. It can be seen that in Figure 10.10 (with respect to Figure 10.7), the center of gravity of Cluster 1 moved in the $Boiler_{level}$ axis and its feature range has grown to cover the zones previously covered by Cluster 5. The center of gravity of Cluster

3 has diminish in the $Boiler_{level}$ axis.

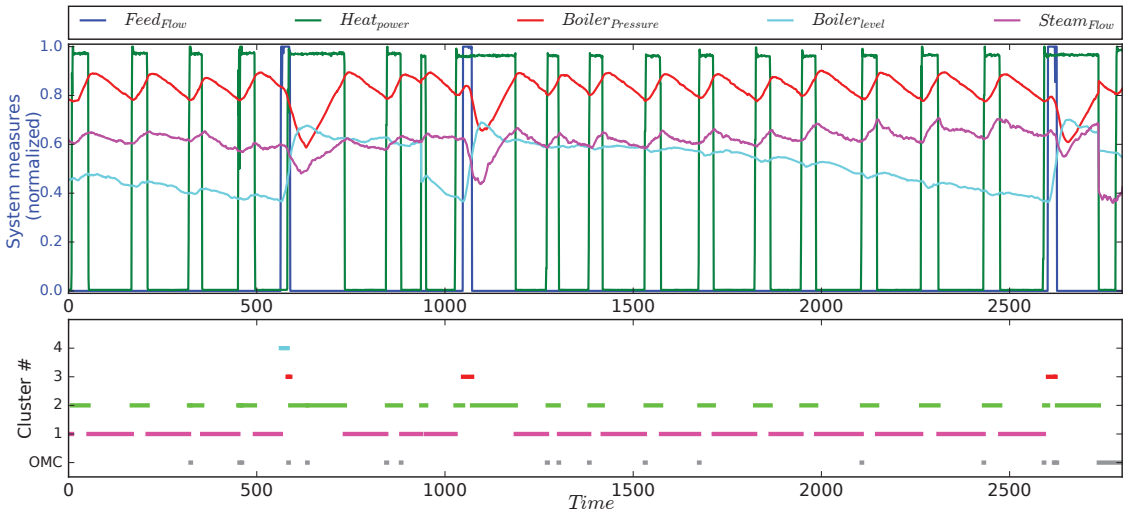


Figure 10.9: Updated ***DyClee*** clustering results of steam generator data in normal operation mode (samples $0 - 2700$). Labels update is performed automatically by the algorithm.

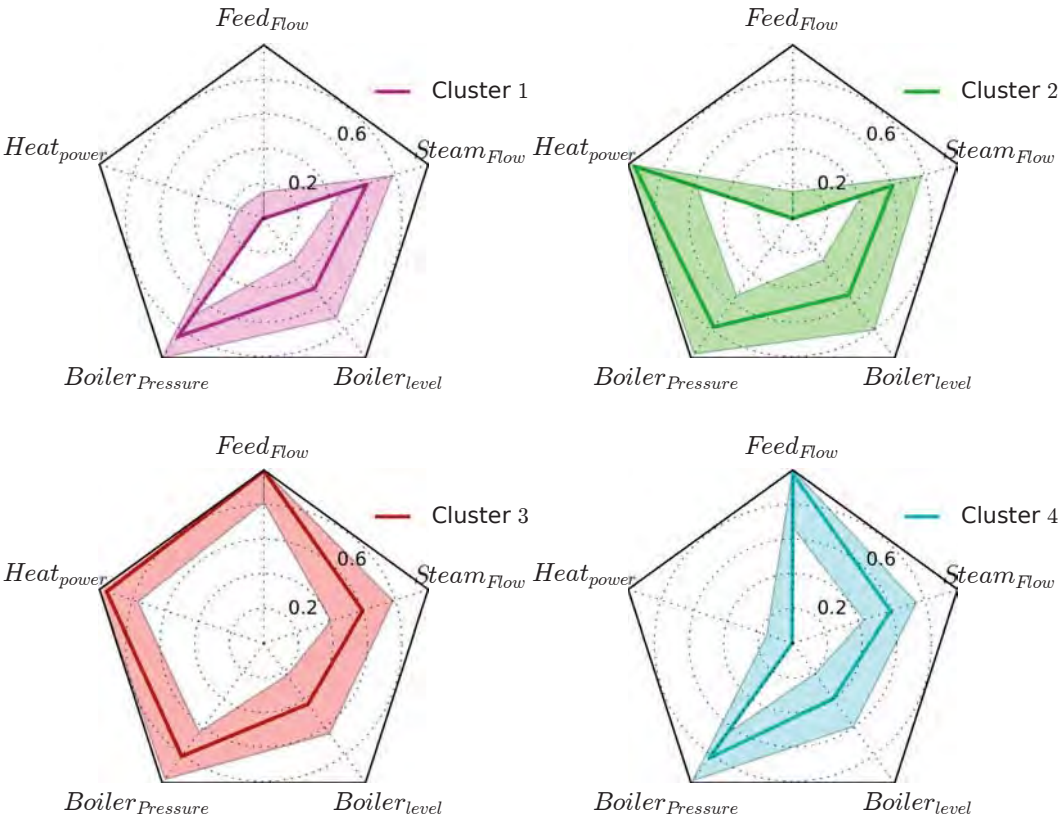


Figure 10.10: Radar representation of the clusters found by ***DyClee***. The thick line represents the cluster center of gravity while the shadow area represents the whole range of the cluster.

To see the cluster merge in detail the μ-clusters representation can be useful. Figure 10.11 shows four snapshots of ***DyClee*** μ-clusters distributions at $t = \{1100, 1700, 2100, 2700\}$. At snapshot $t =$

1700 can be evidenced how clusters 1 and 5 start to getting closer. This rapprochement ends with the clusters merging at $t = 1800$ which is reflected in the $t = 2100$ snapshot. Note also that snapshots $t = 1700$ and $t = 2100$ show no μ-cluster belonging to the Cluster 3. The reason for that is that only those μ-clusters having been updated in the last $3 * t_{global}$ seconds are considered for the analysis. Nevertheless, these μ-clusters remain in the long term memory so that whenever the behavior they represent arises again, it can be recognized immediately, as can be seen in snapshot $t = 2700$. No data were assigned to Cluster 4 since $t = 581$.

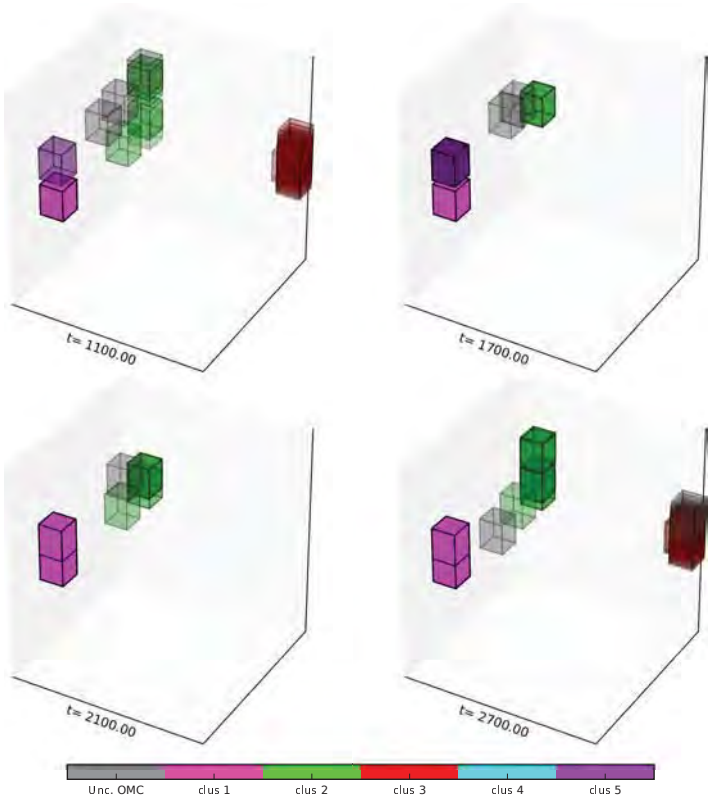


Figure 10.11: ***DyClee*** μ-cluster clustering results of steam generator data in normal operation mode (samples 0 to 2700)

The final ***DyClee*** clustering results are shown in Figure 10.12. In the last 2000 samples ***DyClee*** recognizes the no regulation, pressure regulation and level regulation modes as well as a *new detected behavior*, characterized as Cluster 6 and corresponding to a fault. The cluster's profiles are shown in Figure 10.13. It can be seen that Cluster 6 is characterized by low to zero values in the $Heat_{power}$, $Feed_{Flow}$ and $Steam_{Flow}$ axis with high $Boiler_{Pressure}$. It is worth noting that some samples along the whole experiment were clustered as $\mathbb{O}\mu$Cs. These samples, depicted in gray, represent noise and transitional behaviors.

Between samples 4039 and 4048 the system is in level and pressure regulation mode, nevertheless ***DyClee*** clusters those samples as unknown behavior. To understand this result, Figure 10.14 shows data and clustering results from the interval $t = [4000, 4100]$ as well as the last interval in which both level and pressure were regulated, that is, $t = [2550, 2680]$. It can be seen that the steam output level between the behavior characterized in Cluster 3 and the one registered in the interval $t = [4000, 4100]$

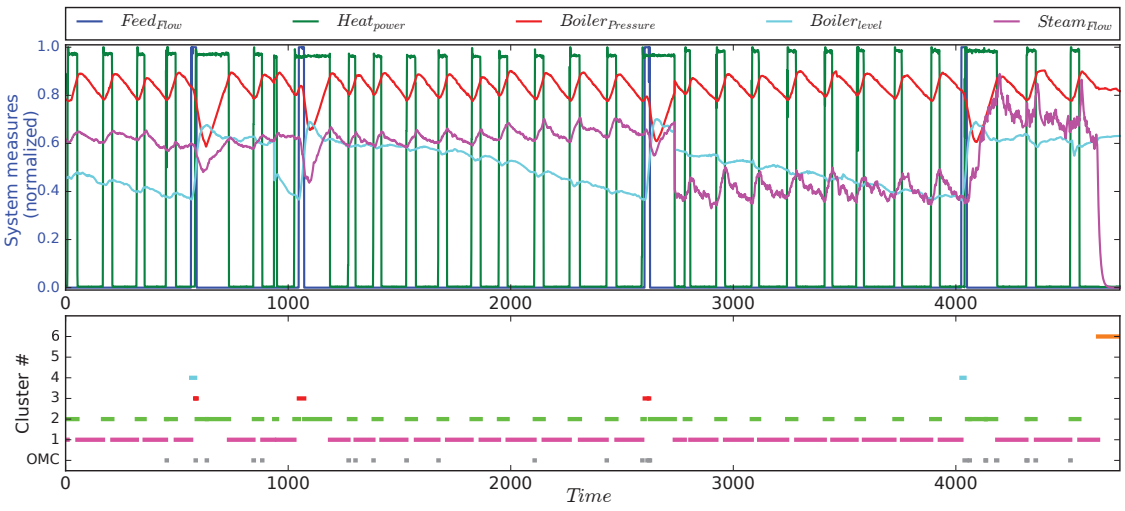


Figure 10.12: ***DyClee*** clustering results of steam generator data

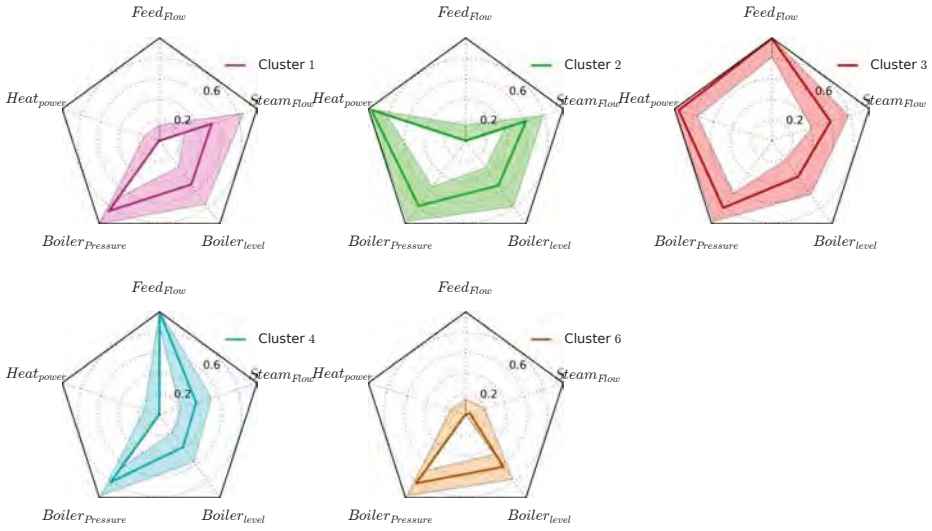


Figure 10.13: Radar representation of the clusters found by ***DyClee***. The thick line represents the cluster center of gravity while the shadow area represents the whole range of the cluster.

is of 0.3 of the feature range, which makes impossible for ***DyClee*** to recognize it. It should be kept in mind that, if in this operational mode the steam level can change between 0.3 and 0.65 of the normalized feature range, future data might connect these behaviors in Cluster 3.

Figure 10.15 illustrate μ-cluster's distribution in snapshots taken at $t = 3000$, $t = 3500$, $t = 4100$ and $t = 4700$. In snapshots $t = 3000$ and 4100 it can be seen that the found μ-clusters cover almost the same space in the plotted dimensions ($Feed_{Flow}$, $Heat_{power}$, $Boiler_{level}$) which confirms that the main difference is in a unplotted dimension. The same conclusion apply for Cluster 6 that share the same ranges that Cluster 1 in the plotted dimensions, as can be confirmed by the clusters profiles in Figure 10.13.

This test case allowed to show ***DyClee*** clustering capabilities. The detection and further merge of

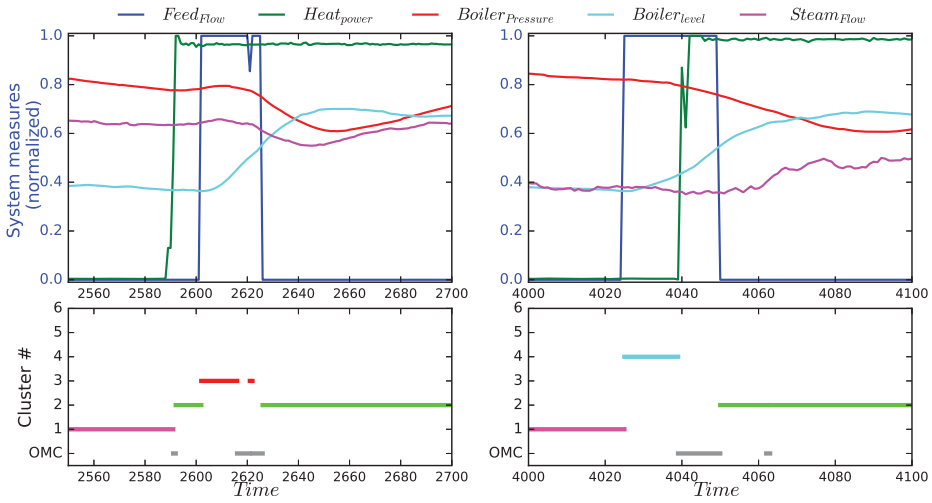


Figure 10.14: Comparison of the system outputs between $t = [2550, 2680]$ ans $t = [4000, 4100]$. On each interval the regulation of both level and pressure is activated, nevertheless the steam output flow differs greatly between these intervals.

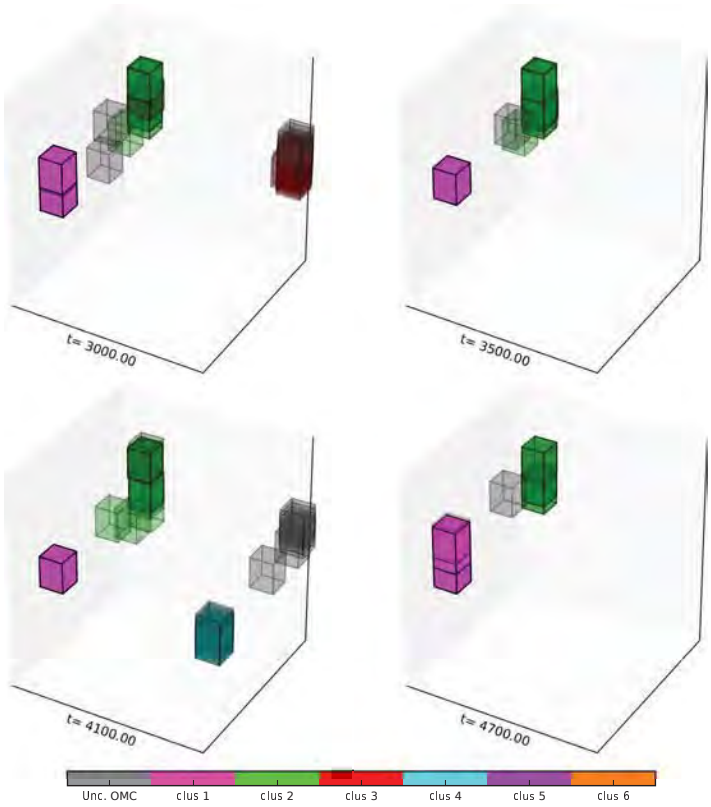
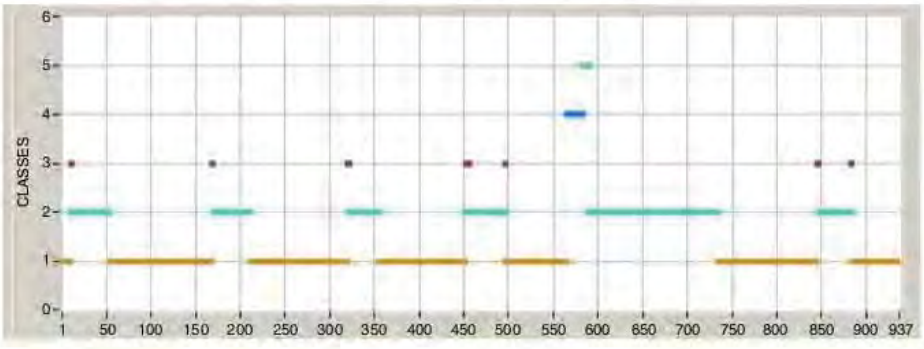


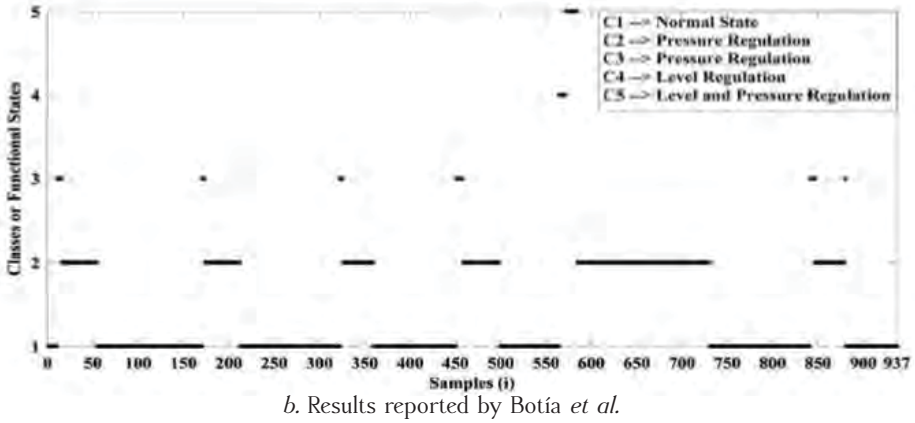
Figure 10.15: ***DyClee*** μ-cluster clustering results of steam generator data. (Snapshots at $t = \{3000, 3500, 4100, 4700\}$)

clusters 1 and 5 are a perfect example of ***DyClee*** robustness to the order of appearance of the provided samples. Results reported in [Kem *et al.* 06] and [Bot *et al.* 13], and illustrated in Figure 10.16, use two clusters to represent the pressure regulation mode while ***DyClee*** represents this behavior in only

one cluster conformed by several μ-clusters along a large influence zone given by the boiler level and boiler pressure variations. ***DyClee*** surpasses the mentioned approaches thanks to its dynamic nature. The availability of μ-clusters of different densities allow to characterize new behaviors without the need of a specific learning stage. The $\mathbb{O}\mu$Cs also allow to describe noise or transitional behavior that should not be assigned to any cluster or it might bias the cluster characterization. The possibility of merge or split classes automatically gives ***DyClee*** robustness against unknown or unregistered data.



a. Results reported by Kempowsky *et al.*



b. Results reported by Botía *et al.*

Figure 10.16: Clustering results in [Kem *et al.* 06] (top) and [Bot *et al.* 13] (bottom) for steam generator first 937 samples. These samples were used as training set in those works.

10.1.2 Discrete event modeling to support the decision making process

Chapter 9 introduced the formalism to use ***DyClee*** clustering results to automatically build a timed automaton. In this subsection ***DyClee*** results, obtained on the steam generator, are used to build such DES model. In the sake of clarity these results are shown for the three time-spans analyzed in subsection 10.1.2.

The timed automaton at $t = 999$, corresponding to the clustering results of Figure 10.6, is depicted in Figure 10.17. The five states corresponding to ***DyClee*** natural clusters are depicted as well as the complementary states denoted as $\sim i$. System events are not depicted explicitly but each transition from a Cl_i state to a $\sim i$ state implies the $\overline{Clus_i}$ event and each transition from a $\sim i$ state to a Cl_j state implies a $Clus_j$ event. Each cluster state includes, below the cluster identification, the information related to the median time that the system had remained in each of these states. Clocks constraints (in

Cl_j to $\sim i$ transitions) and resets (in Cl_j to $\sim i$ transitions) are included in the graph according to the notation explained in Chapter 9. The probability of a transition is represented between parenthesis at the beginning of each transition for the $\sim i$ states. For the Cl_i states the only allowed transition is to $\sim i$, fired by the $\overline{Clus_i}$ event. Since this transition has a 100% probability of occurrence it is not depicted in the graph. The label of the current state is depicted in red.

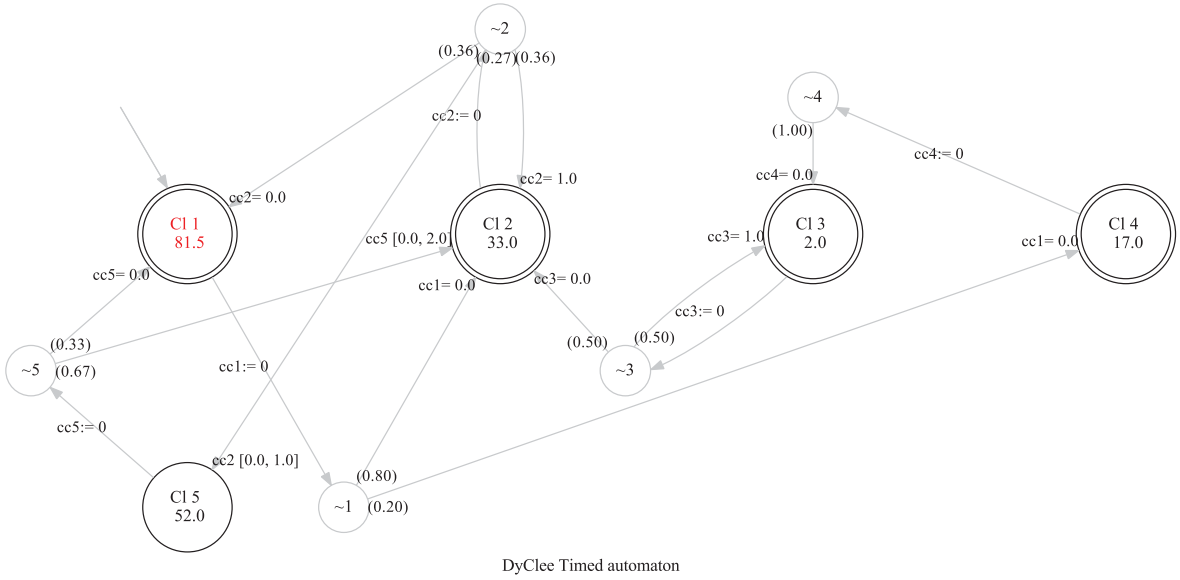


Figure 10.17: ***DyClee*** timed automaton generated automatically from clustering results of steam generator data (samples 0 to 999)

The graphic representation of the timed automaton at $t = 2700$ is depicted in Figure 10.18. The states corresponding to Cluster 5 have disappeared but as informative measure the name of the merged clusters is kept in the diagram ($m : \{5\}$). In this case the prevailing cluster label changes from black to blue illustrating that another cluster merged with it, and the merged cluster label is added to the merged list that is now shown at the bottom of the state label. It can be seen that the timed constraints of Cluster 1 now reflect also the information previously contained in Cluster 5. At $t = 2700$ the current state is ~ 2 reflecting that the system is in transitional state leaving Cluster 2. The possible destinations are clusters 1 and 3. The label of the represented operational modes were added to Figure 10.18 in order to facilitate its comparison with the results found in [Kem *et al.* 06] and depicted in Figure 10.19. It can be seen that the automaton found in [Kem *et al.* 06] recognize only the transitions found in the training stage which forces the expert to add transitions to explain observed behavior (see the dotted line transition between $C2$ and $C5$ in Figure 10.19). ***DyClee*** updates the timed automaton automatically using the information about novelties reported by the clusterer. For example, in Figure 10.18 the transition between the pressure regulation mode (Cl_2) and the level and pressure regulation mode (Cl_3) are added automatically.

The diagram of the timed automaton at the end of the test is shown in Figure 10.20. This diagram depicts the creation of a new state (Cl_6) and informs that the system has been in that state for 97 samples. Diagrams in figures 10.17 and 10.20 show four states in double circled nodes and one state in a single circle node. The double-circled notation reflects the set of marked states, that is, the

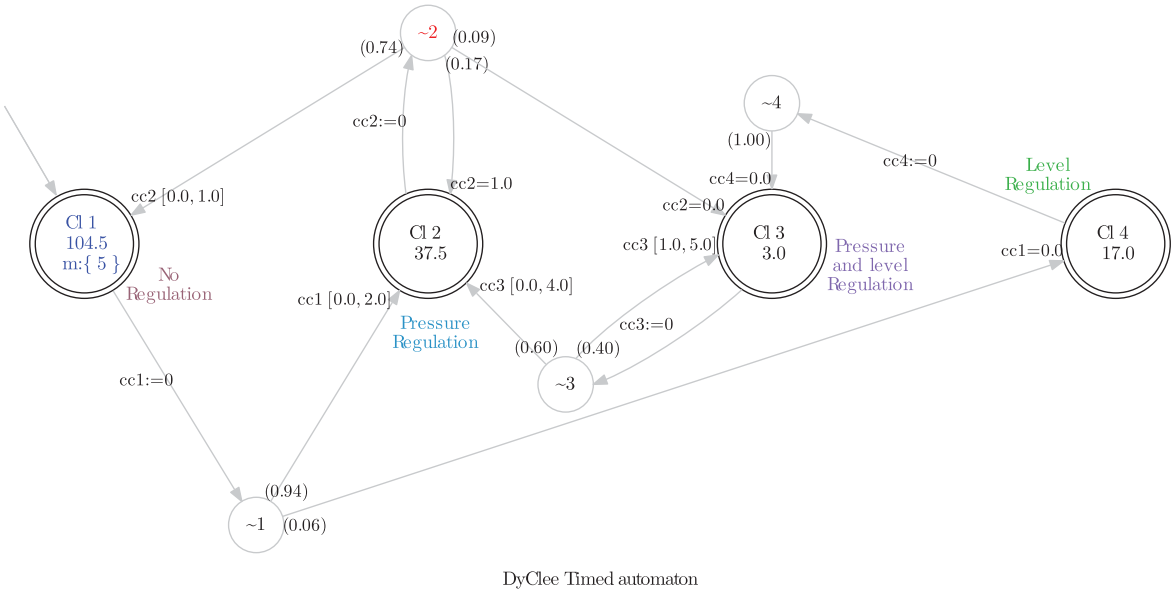


Figure 10.18: ***DyClee*** timed automaton generated automatically from clustering results of steam generator data (samples 0 to 2700)

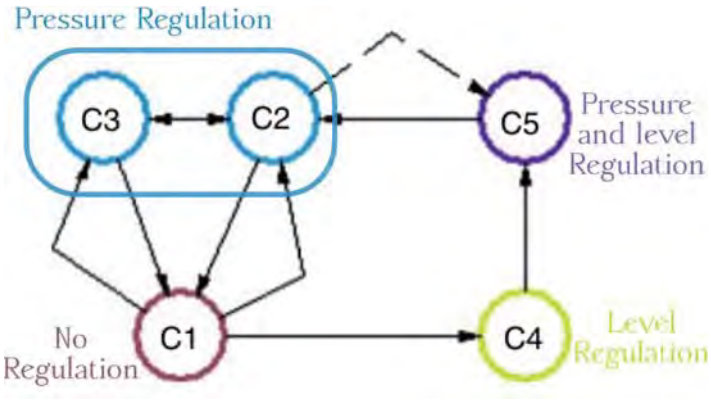


Figure 10.19: Automaton found in [Kem *et al.* 06] continuous lines depict transitions found by their approach, the dotted transition is added by the process expert.

set of states that ***DyClee*** considers as valid operational states. This set is found dynamically in an incremental fashion using as only external parameter the number of operational modes (provided by the process expert).

10.2 The continuous stirred tank heater (CSTH)

The CSTH is a benchmark of a stirred tank in which hot (50°) and cold (24°) water are mixed and further heated using steam; the final mix is then drained using a long pipe. The configuration of this benchmark, developed by [Tho *et al.* 08], is shown in Figure 10.21. It is assumed that the tank is well mixed so the temperature of the outflow is the same as that in the tank. Process inputs are set-points for the cold water, hot water and steam valves. Process outputs are hot and cold water flow, tank

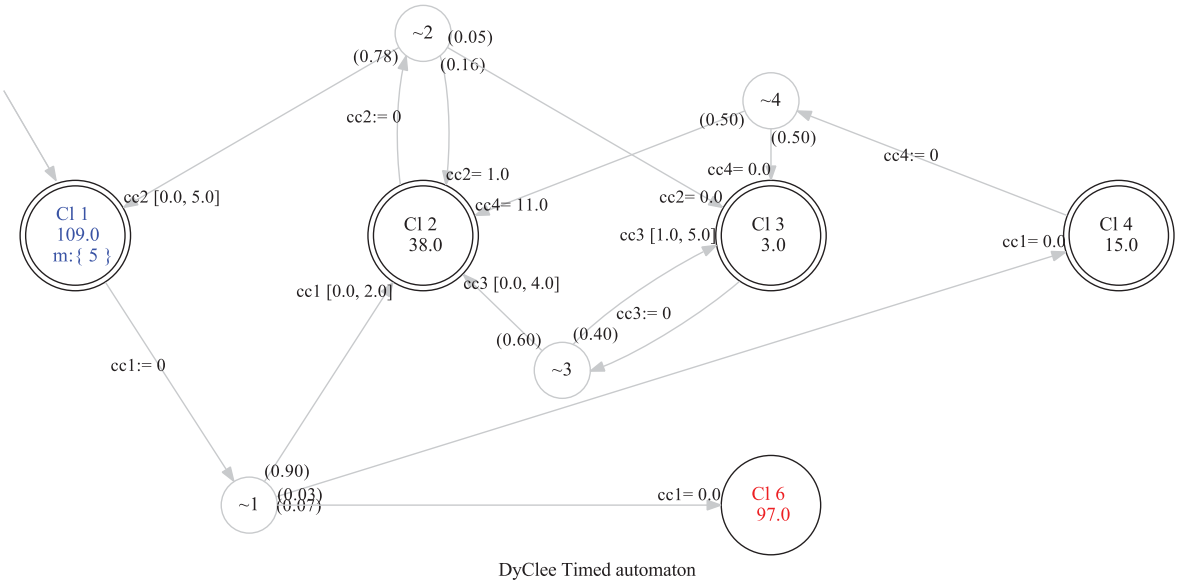


Figure 10.20: ***DyClee*** timed automaton generated automatically from clustering results of steam generator data (whole data set)

level and temperature. Process inputs and outputs represent electronic signals in the range $4-20mA$. The benchmark is tested in closed-loop. PID controllers are used to guide the plant as suggested in [Tho *et al.* 08].

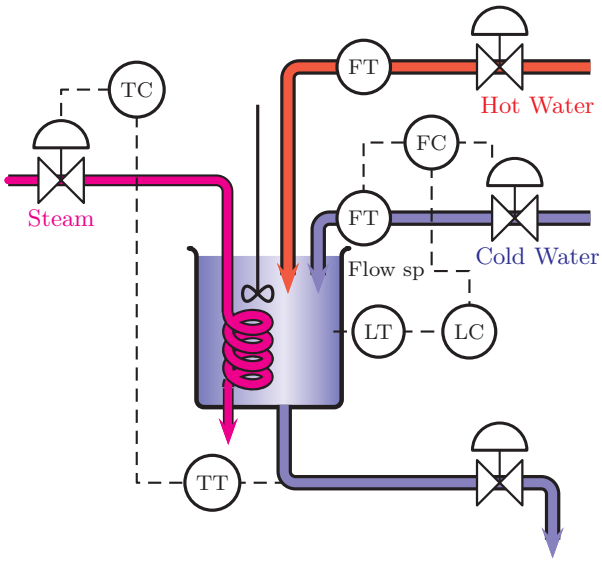


Figure 10.21: The continuous stirred tank heater

Thornhill *et al.* in [Tho *et al.* 08] suggests two operation points depending on whether or not the hot water flow is used. The suggested set-points for the operation points OP1 and OP2 are shown in table 10.3. Simulink models are available at the website [Tho 15], with and without disturbances. The provided disturbance signals are real data sequences experimentally measured on the pilot plant at the University of Alberta. This benchmark does not implement directly process faults but since

it implements the process dynamics crearly the fault can be added by including directly their dynamic. The modifications made on the original [Tho 15] Simulink files are presented below. These modifications include an interface that allow to interact with the faults easily.

Variable	$OP1$	$OP2$
$Level(mA)$	12.00	12.00
$Level(cm)$	20.48	20.48
$CWflow(mA)$	11.89	7.330
$CWflow(m^3s^{-1})$	9.038×10^{-5}	3.823×10^{-5}
$CWvalve(mA)$	12.96	7.704
$Temperature(mA)$	10.50	10.50
$Temperature(°)$	42.52	42.52
$Steamvalve(mA)$	12.57	6.053
$HWvalve(mA)$	0	5.500
$HWflow(m^3s^{-1})$	0	5.215×10^{-5}

Table 10.3: Suggested operational points for the CSTH

The Simulink macro model is show in Figure 10.22. The set-points for the process are set in milliamperes. The plant control is based in three set-points: the desired level (level sp/mA), the desired temperature (temp sp/mA) and the hot water valve position (HM valve/ma). Process disturbances can be set on/off by modifying the value of the product block to 1/0.

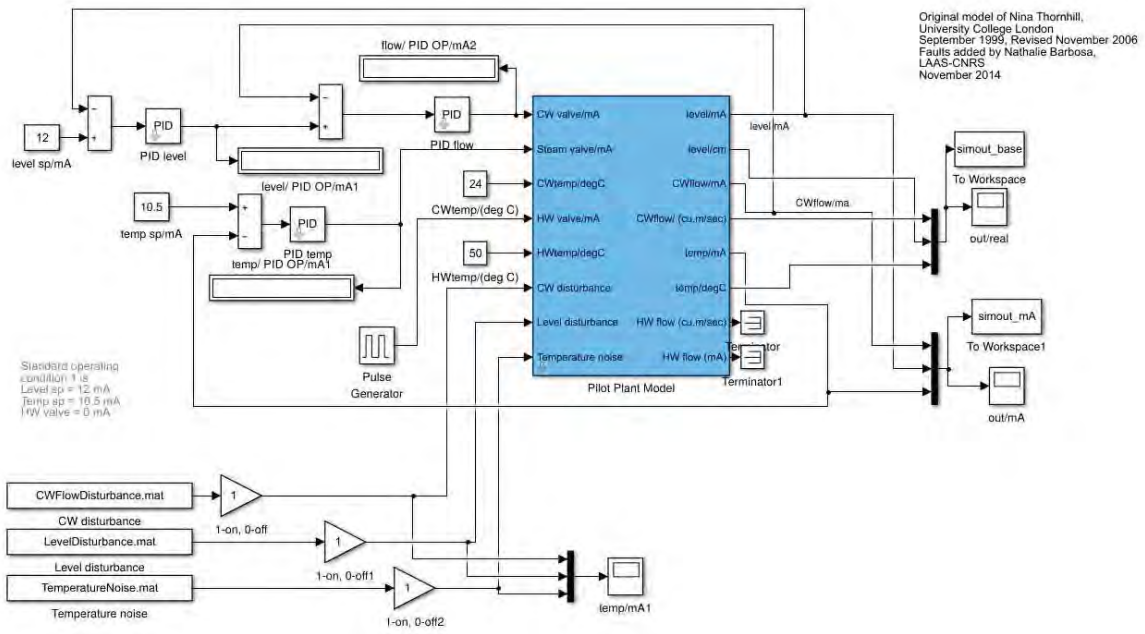


Figure 10.22: Simulink model of the continuous stirred tank heater

To simulate faulty scenarios a set of faults was included inside the process dynamics and can be activated through the main block mask which is shown in Figure 10.23. The first field activates/deactivates evolving leaks. Second and third fields allow to simulate a blocking condition in the hot water valve, the former indicated blocking percentage of the valve and the later activate or deactivate the fault. Forth and fifth fields allow to simulate a blocking condition in the steam valve. These fields correspond to the blocking percentage and the fault activation respectively.

Figure 10.23: Pilot Plant Model Mask. It allows to activate or deactivate the implemented faults for change the simulation scenario.

To understand the benchmark implementation and how the modification where made it is necessary to see the internal blocks of the 'Pilot Plant Model'. Four blocks describe the plant behavior and interactions, the first block describe the cold water dynamics, in second block the mass balance is calculated, the third block describe the hot valve dynamics and in the forth block the heat balance is calculated. These blocks are depicted in Figure 10.24.

The dynamics of the cold water valve were modeled by Tornhill *et al.* according with the data of the pilot plant in the University of Alberta. The flow disturbances are included in the valve dynamics as additive behavior as seen in Figure 10.25. The hot water valve block was modified to emulate a the blocking of the valve. This modification was implemented using a switch as can be seen in Figure 10.26. The valve reference value is set to the stuck value[1] if the fault is active and to the set-point otherwise. The activation time and duration of the fault can be customized by using the signal builder block (red block in the figure). Possible faults include permanent valve stuck and intermittent valve stuck.

The mass balance is calculated in the block shown in Figure 10.27. In order to include leaking faults a block was added to the original model. This block, depicted in red in Figure 10.27 estimates the flow due to fluid leakage. The fluid loss due to leaks is in nature incremental since the radius of the hole causing the leak tends to grow as time passes due to the force exerted by the leaking fluid.

[1]The valve stuck value is specified as a percentage in the global fault management window found by double-clicking in the Pilot plant model block

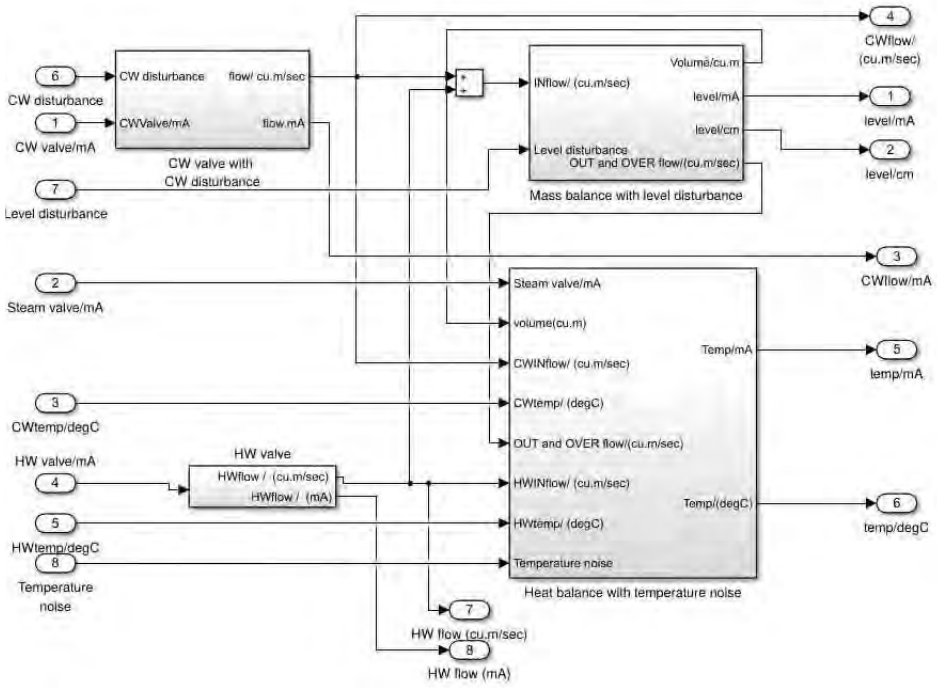


Figure 10.24: Bock description of the Pilot Plant Model

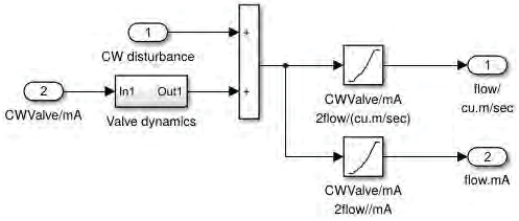


Figure 10.25: Cold water valve dynamics

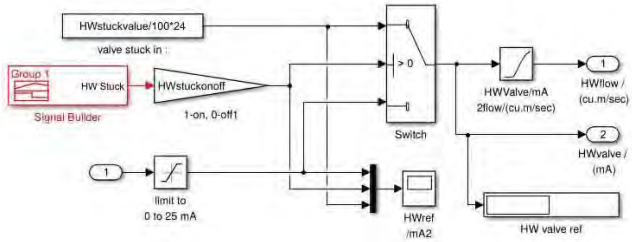


Figure 10.26: Hot water valve dynamics including stuck valve fault

Leaks are considered as evolving (incremental) faults following the dynamics described in equation 10.1, where θ_c is the discharge coefficient, $r(t)$ represent the hole radius, g the gravity force and h_t the tank level. Figure 10.28 shows how the leak flow is calculated.

$$Flow_{leak} = \theta_c \pi (r(t))^2 \sqrt[2]{2 * g * h_t} \tag{10.1}$$

Each one of the 'evolving leak' blocks showed in Figure 10.28 implement a leak. The radius of the leaking hole $r(t)$ starts growing at the start time and grows at a rate equals to $radius = \frac{\text{growing slope}}{\text{Time of evolution}}$ during the time of evolution. The radius then remains constant until the leak end time. The leak parameters can be customized in the evolving leak parameters window[2]. A view of this window is presented in Figure 10.29. This implementation allow to simulate scenarios of multiple incremental leaks with different dynamics occurring at different times but also overlapping faults.

[2]This window is accessible by double-clicking in the evolving leak box.

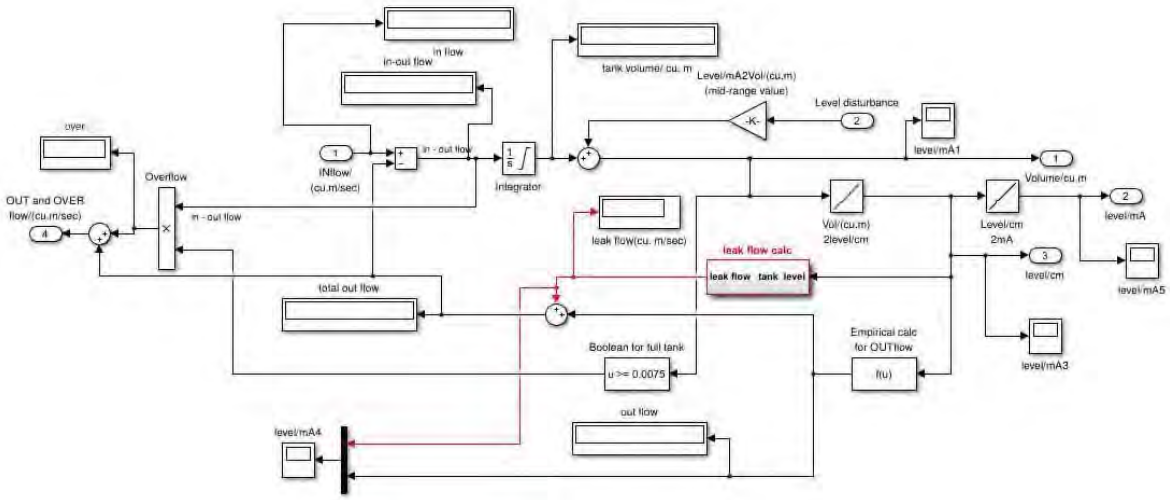


Figure 10.27: The Mass Balance block: The system mass balance is calculated inside this block

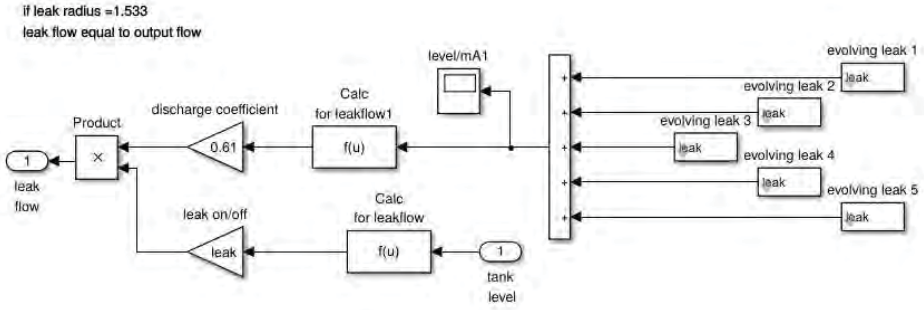


Figure 10.28: Implementation of leaks inside the model

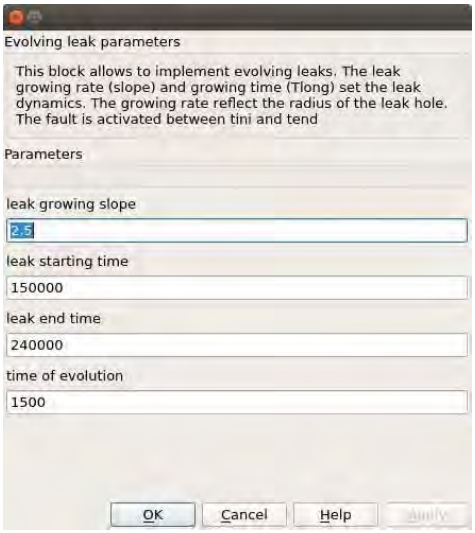


Figure 10.29: Evolving leak block parameters: Mask view

The heat balance is calculated in the block shown in Figure 10.30. In this block a fault was introduced to block the steam valve. The fault implementation is the same as the one implemented

for the hot water valve. The starting time and percentage of blocking are set in the global fault management window depicted in Figure 10.23.

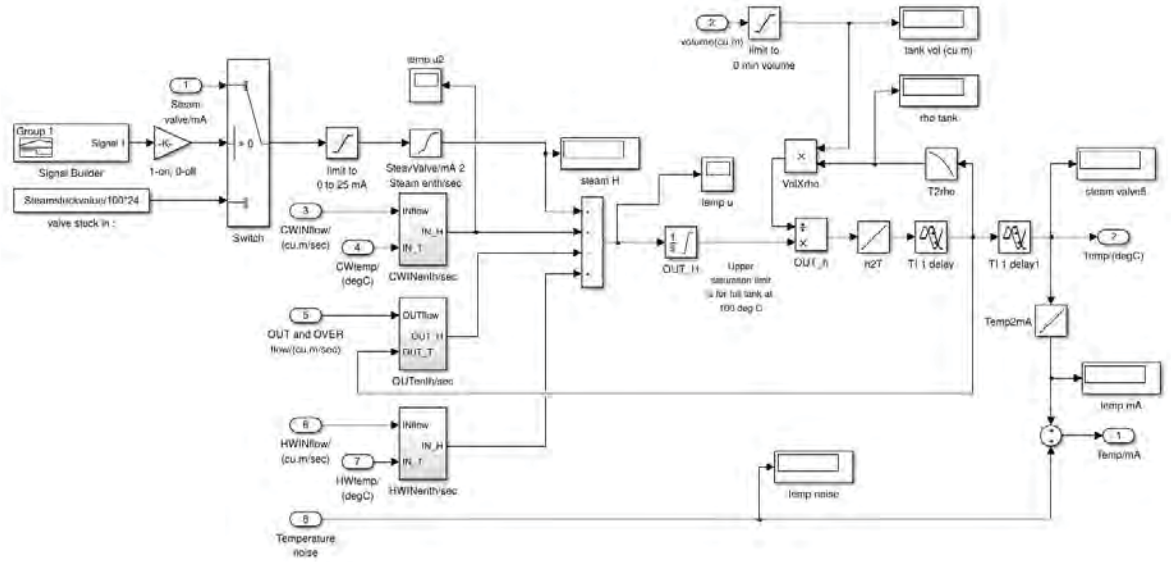


Figure 10.30: The heat balance block

10.2.1 Monitoring different operational scenarios

Drift in the operational point as well as tank leakage and stuck valve scenarios were simulated in Simulink, including the given disturbances, and the resulting data were fed to ***DyClee***. The following experiments use (unless otherwise stated) the user defined parameters and values shown in Table 10.4. Test results are shown as snapshots of the μ-cluster distributions at different points in the simulation. The μ-clusters are represented as cubes centered in the μ-cluster center. For the x, y and z axes, the three features with maximal variance are chosen in order to ease graphical interpretation of the results. The cube color represents the μ-cluster class and the μ-cluster density is represented as the cube opacity.

Parameter	`forget_method`	$t_{w=0}$	`ws`	`Min_ws`	φ	t_{global}	ρ	ζ
Value	linear	30`ws`	600 (5 min)	120 (1 min)	1	300 sec	2.0	1.6

Table 10.4: Parameters used in the experiment

Scenario 1: Tracking a drift in $OP1$ due to wearing

One common problem in industrial applications is that states might drift when the physical parts of the system are exposed to wearing processes. In the case of the CSTH, we simulate the evolution of $OP1$ when residues accumulate in the border of the output pipe causing an inconsiderable drop in the output flow. This wearing process must be considered normal, meaning that the operation state remains $OP1$ although it suffers a drift.

The process measurements for Scenario 1 can be seen in Figure 10.31. ***DyClee*** clustering results are shown in figures 10.32, 10.33 and 10.34. The extracted trends are shown in Figure 10.32 (top) in dashed lines over the original signals plotted as continuous lines. This figure also shows the clustering results of ***DyClee*** at the bottom reported by black horizontal lines indicating the cluster number according to the scale on the left.

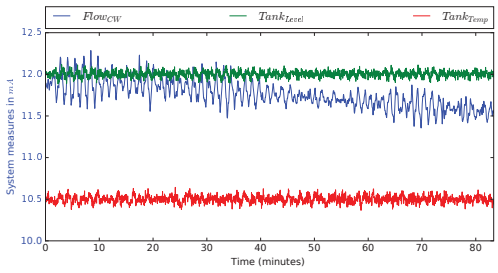


Figure 10.31: System measures for the $OP1$ drift scenario

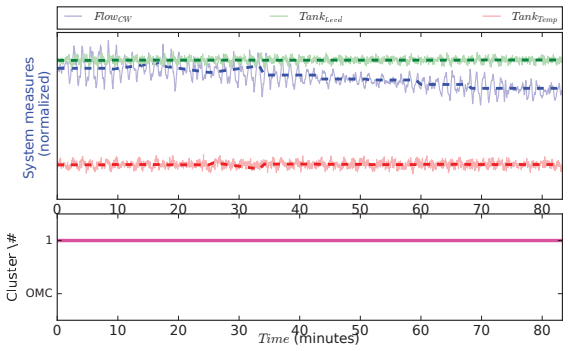


Figure 10.32: Dynamic Classification of the CSTH $OP1$ drift scenario. At the top: Trends (dashed lines) over normalized system measures (continuous lines). At the bottom: clustering results

The μ-clusters evolution is shown in Figure 10.33. In spite of disturbances, the stable behavior of the plant at the beginning of the simulation is reflected in a single one class one μ-cluster classification, in accordance with uniform data. The opacity of the μ-cluster represent its density normalized to the maximum density in the analyzed period. In Figure 10.33 the μ-cluster density augment as time passes, since all the episodes are grouped together. Following the system evolution by seeing its measures can be a difficult task and even impossible when dozens of signals are analyzed at the same time. To follow system evolution through the variation in its variables, a radar-like graphic representation is proposed. For the simulated scenario the graphics for $t = 17.07$ minutes and $t = 93.87$ minutes are shown in Figure 10.34. The drift in the $Flow_{CW}$ variable are depicted as the filled area between the original characterization of $OP1$ and its current characterization.

Scenario 2: Tracking an evolving leak in the CSTH

In order to simulate faulty dynamical behavior, an increasing size leak was simulated. CSTH was working on OP1 and its output signals were sampled at $T_s = 0.5s$. A leak caused by a hole was introduced on sample number $n = 3300$ ($t = 27.5$ minutes) as shown in Figure 10.35. The starting radius of the hole is $r = 1mm$. The experiment emulates the case where the size of the hole increases as time passes. The final radius of the hole at $t = 50$ minutes is $r = 3.75mm$. The algorithm start window size was set as `ws`= 512 samples, the minimum size as `Min_ws`= 68 and the maximum as `Max_ws` = 1028. The measured output signals seen in Figure 10.35 were taken as ***DyClee***'s input. The found trends are shown as dashed lines in Figure 10.36 (top). The clustering labels are shown at the bottom of Figure 10.36 using the same time axis.

Figure 10.37 shows four snapshots of ***DyClee*** structure evolution reflected in the drift and creation

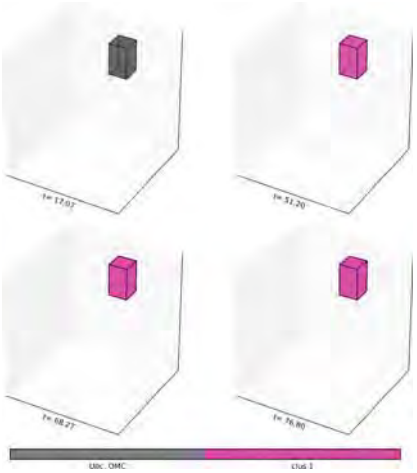


Figure 10.33: μ-clusters evolution for the drift scenario

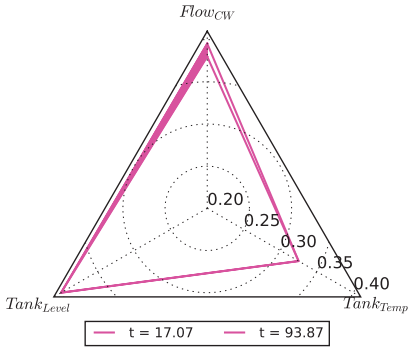


Figure 10.34: Radar representation of found clusters

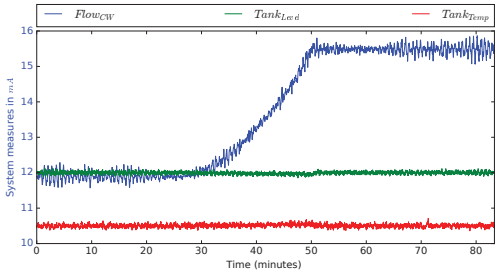


Figure 10.35: System measures for the dynamic leak simulation.

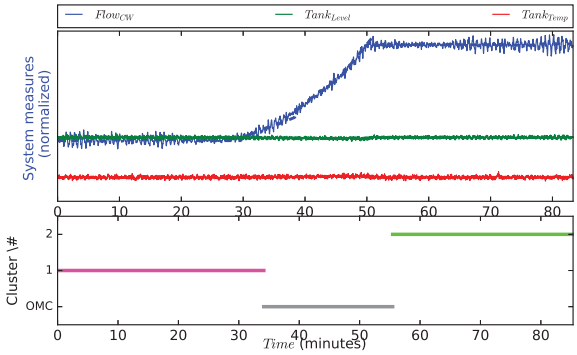


Figure 10.36: Dynamic Classification of the CSTH in scenario 2. At the top: Trends (dashed lines) over normalized system measures (continuous lines). At the bottom: clustering results

of μ-clusters. As explained before, the three features with larger variance were selected as axes in the $3D$ plot. The first snapshot show a single class single cluster configuration representing $OP1$. After leak introduction, other μ-clusters are created out of $Class1$. Transitory behavior is detected in snapshots $t = 51.2$ and $t = 68.27$, reflected by $O\mu$-cluster creation. As soon as the amount of data in the full leak state increases (see the increased density in snapshot $t = 68.27$), the algorithm is able to detect that this transitional behavior has led to a new behavioral state, formally described by $Class2$ (snapshot $t = 89.6$). In this case the new state corresponds to the faulty state induced by the leak.

The beginning of the fault can then be tracked to the moment in which the system leaves $OP1$, i.e. the creation time of the first $O\mu$-cluster in the frame, which correspond to $t = 29.8$ minutes. Figure 10.38 shows the characterized natural clusters using the radial feature representation. This representation shows that the cold water inflow is the key feature to distinguish the leak faulty behavior. This example illustrates the ability of ***DyClee*** to characterize and remember functional states and at the same time forgetting those transitory behaviors in which the system may not return.

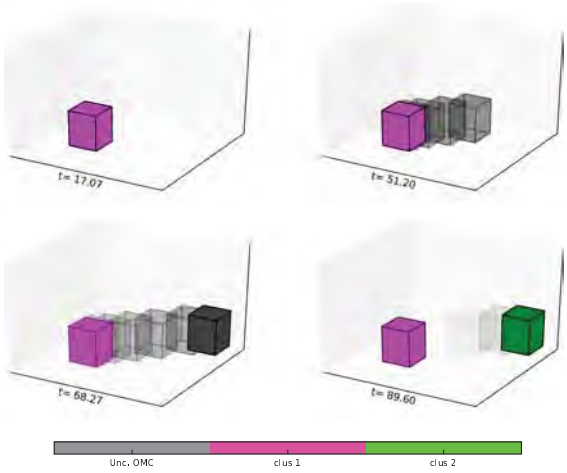


Figure 10.37: μ-clusters evolution for the evolving leak scenario

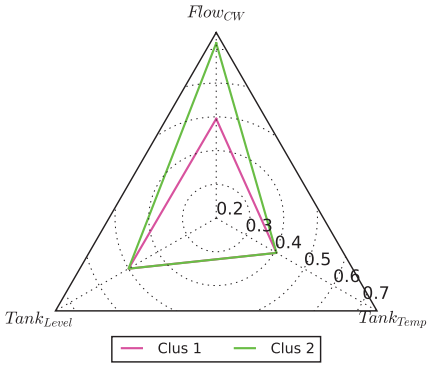


Figure 10.38: Radar representation of found clusters

Scenario 3: Tracking a stuck valve in the CSTH

Valve failure detection is also tested. Figure 10.39 shows the CSTH outputs when the cold water inflow valve is blocked in 60% of its capacity at $t = 16.6$ minutes, with signals sampled at $T_s = 0.1s$. CSTH was working in OP1. For this experiment the algorithm start window size was established as `ws`= 1024 samples, `Min_ws`= 512 and `Max_ws`= 4096. The linear function was chosen as forgetting function and $t_{w=0}$ was set to 80 minutes. Trends representing the system behavior are shown at the top Figure 10.40 as dashed lines. The cluster labels are shown at the bottom of Figure 10.40 using the same time axis.

Figure 10.41 shows snapshots of ***DyClee*** μ-cluster evolution. Snapshot $t = 11.99$ shows the behavior of the process in $OP1$. Snapshot $t = 23.99$ display an increasing amount of $O\mu$-clusters, some of them will evolve into $D\mu$-clusters representing stuck valve behavior in $Class2$, as can be seen in snapshots with $t \geq 35.98$. The radar representation of the characterized natural clusters is shown in figure 10.42. This representation marks the tank level as key feature for the detection of this fault. Like the previous scenario, this scenario illustrates that transitory behavior is not erroneously detected as class drift by ***DyClee***.

Scenario 4: long-term tracking of multiple non persistent faults

This scenario spans over a long time window and aims at showing that ***DyClee*** is able to detect and memorize behavior characteristic of a given operational state. For this purpose, occurrence and disappearance of several faults, i.e. evolving leaks and stuck valves, were simulated to represent real process life cycles. The total simulation time of this scenario is equivalent to the timespan of a month (2.419.200 *seconds*) in which the plant works half of the time in $OP1$ and the other half in $OP2$ (operational points described in Table 10.3). The faulty events included in this scenario are detailed in Table 10.5 with their date of occurrence. Notice that faults occur and are repaired later like in the real life process. Faults are hence non persistent and the difficulty of this scenario is to manage novelty detection while also recognizing previously detected classes.

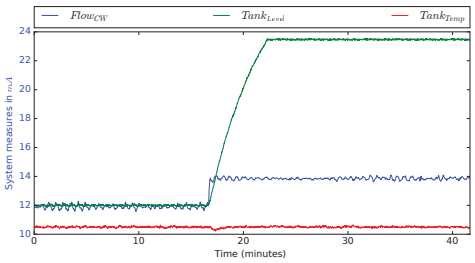


Figure 10.39: Simulation of valve stuck in 60% at $t = 1000$.

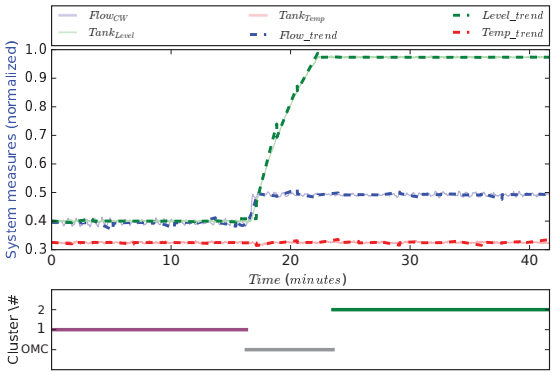


Figure 10.40: Dynamic Clustering of the CSTH for valve stuck detection. At the top: Trends (dashed lines) over normalized system measures (continuous lines). At the bottom: clustering results

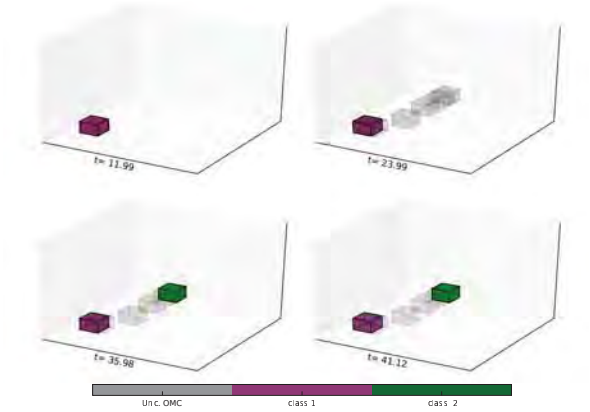


Figure 10.41: μ-clusters evolution for the valve stuck scenario

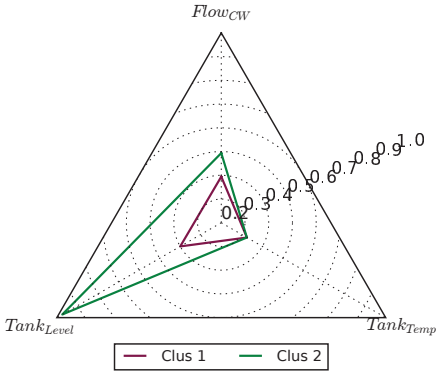


Figure 10.42: Radar representation of found clusters

CSTH output signals are shown in Figure 10.43 together with tags illustrating the event occurrence up arrows indicate the event introduction while down arrows inform about the system reparation or change in the operational mode. At the beginning of the simulation the CSTH was working in $OP1$. Then, faults are injected and repaired as scheduled in Table 10.5, resulting in 9 operational states shown in the bottom of Figure 10.44. This figure also depict the found trends in dotted lines. As in the previous simulations only process outputs were used as input for the clusterer.

As seen in Figure 10.44, ***DyClee*** can successfully track online the process and its evolution. Figures 10.45 to 10.49 show ***DyClee*** μ-clusters evolution and an interesting tool to analyze this simulation. For this experiment the sampling time is $T_s = 0.5s$. The algorithm start window size set to `ws`= 5000 samples (2500 seconds), `Min_ws`= 1000 samples and `Max_ws`= 10000 samples. The forgetting process was activated and the 'linear' forgetting function was selected with $t_{w=0} = 1.50 \times 10^5$ seconds.

In Figure 10.45 μ-clusters evolution between $t = 0.2 \times 10^5$ seconds and $t = 3.1 \times 10^5$ seconds is shown. It is worth remembering that ***DyClee*** learns at the same time as it clusters the data (online). In other words, ***DyClee*** requires no training stage. The normal $OP1$ behavior is detected as soon as

Time (sec.)	Event	Description
0	$OP1$	Plant simulation Start at OP1
150000	l_1	Evolving leak starts. Hole diameter goes from 1 to $3,5mm$ in 1500 seconds
240000	$\bar{l_1}$	Leak fixed
350000	l_1	Evolving leak starts. Hole goes from 1 to $3,5mm$ in 1500 seconds
380000	l_2	A second evolving leak starts. The second hole goes from 0 to $1mm$ in 1500 seconds
450000	$\bar{l_1}\bar{l_2}$	Leaks fixed
550000	s_1	Steam Valve stuck (closed)
578000	$\bar{s_1}$	Valve repaired
650000	s_2	Hot water valve stuck at 10%
700400	$\bar{s_2}$	Valve repaired
900000	l_3	Evolving leak starts. Hole goes from 1 to $2,6mm$ in 1000 seconds
964800	$\bar{l_3}$	Leak fixed
1200000	OP 2	Plant changed to OP2
1500000	s_3	Steam valve stuck at 10%
1557600	$\bar{s_3}$	Valve repaired
1800000	l_4	Evolving leak starts. Hole goes from 1 to $3mm$ in 3000 seconds
1840000	$\bar{l_4}$	Leak fixed
2000000	s_4	Hot water valve stuck at 40%
2061000	$\bar{s_4}$	Valve repaired

Table 10.5: Multiple fault simulation over a month timespan

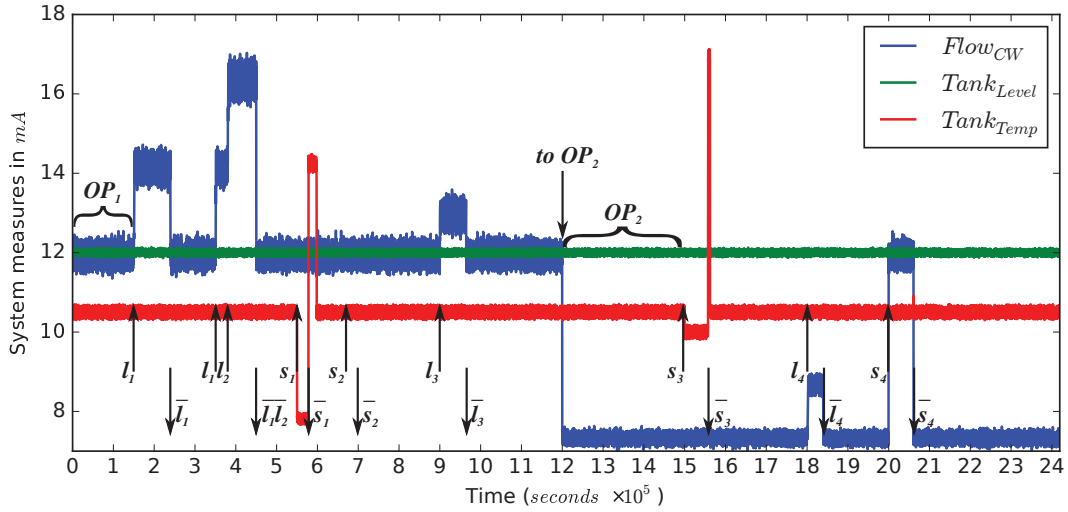


Figure 10.43: Process measurements for multiple fault scenario

$t = 0.20 \times 10^5$ and confirmed with the information arriving between $t = 0$ and $t = 1.50 \times 10^5$. This information allows ***DyClee*** to improve the description of the plant in normal operation point $OP1$. This behavior is labeled as $Cluster1$. The first fault is introduced at $t = 1.50 \times 10^5$. An evolving leak with a hole starting at a radius of $r = 1mm$ and increasing as time passes was simulated. The final radius of the hole is $r = 3.5mm$ at $t = 1.52 \times 10^5$. Snapshot at $t = 1.8 \times 10^5$ shows ***DyClee***'s response to process evolution, the creation of $Cluster2$. New μ-clusters were created for tracking this evolution. At $t = 2.40 \times 10^5$ the leak is fixed and the process returns to $OP1$ normal behavior ($Cluster1$). At this point the reader can see in the snapshots how the density of the μ-clusters increases when new elements are assigned to the μ-cluster (e.g. opacity change in $Cluster1$ from snapshot $t = 0.5 \times 10^5$

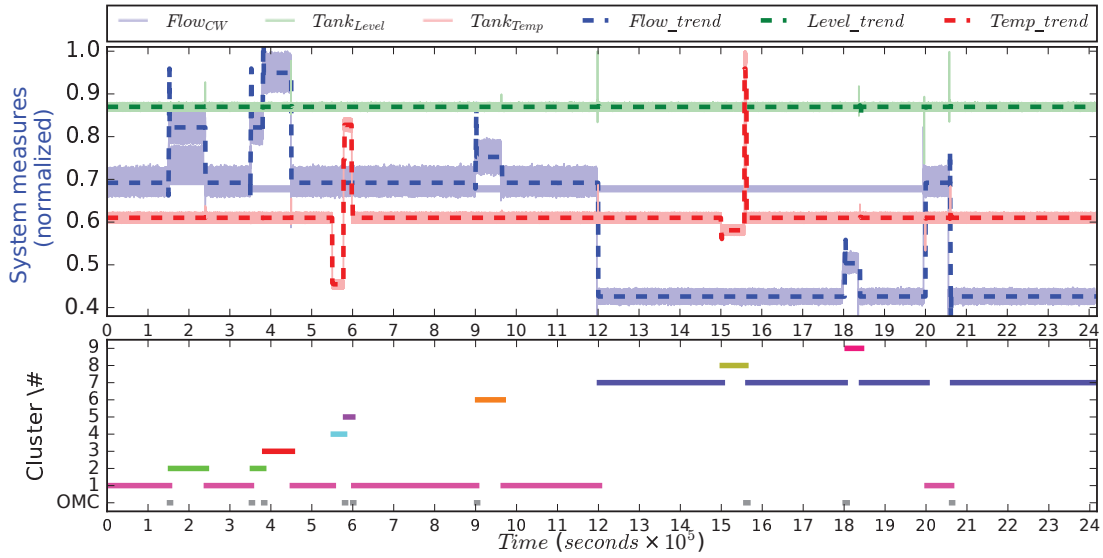


Figure 10.44: Dynamic Clustering of the CSTH for Scenario 4. At the top: Trends (dashed lines) over normalized system measures (continuous lines). At the bottom: clustering results

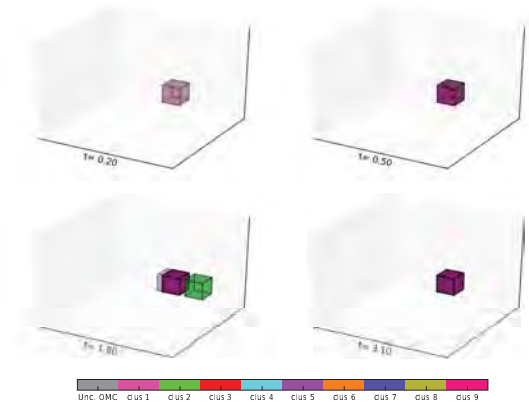


Figure 10.45: ***DyClee***'s μ-clusters evolution for Scenario 4 t between 0.2 and 3.1 seconds $\times 10^5$

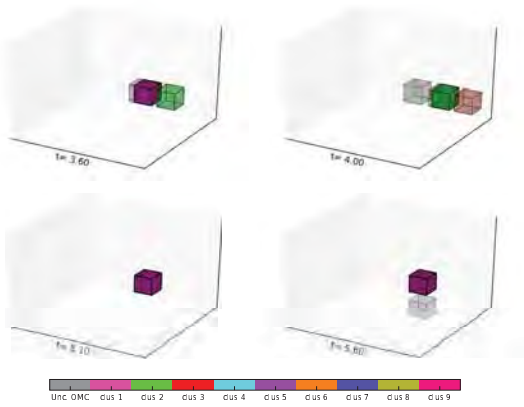


Figure 10.46: ***DyClee***'s μ-clusters evolution for Scenario 4 t between 3.6 and 5.6 seconds $\times 10^5$

to $t = 3.10 \times 10^5$).

At $t = 3.50 \times 10^5$ an evolving leak with the same characteristics as the one at $t = 1.50 \times 10^5$ is introduced. Since this behavior is already known ***DyClee*** recognizes it quickly, turning the output to *Cluster*2 (Figure 10.46). It is worth noting that density of the μ-clusters is plotted in reference to that maximal density achieved by a μ-cluster. This causes faulty $D\mu$-clusters (which are locally dense) to look pale next to the μ-clusters that represent normal behavior.The introduction of a second leak causes the creation of a new μ-cluster that is later characterized as *Cluster*3 as shown in the second snapshot of Figure 10.46. *Cluster*3 density increase arriving to a maximum at $t = 4.50 \times 10^5$. When the leaks are fixed the density of the μ-clusters diminishes, which is represented in the snapshots with less opacity or the remove of the μ-cluster (see snapshot at $t = 5.1 \times 10^5$). It is worth noting that μ-clusters disappear from snapshots when their density downs to zero.

The blockage of the steam valve in its closed position is introduced at $t = 5.50 \times 10^5$. Snapshot at $t = 5.60 \times 10^5$ in Figure 10.46 and Snapshot at $t = 6.0 \times 10^5$ in Figure 10.47 show the tracking of this new behavior which ends in the creation of a new class $Cluster4$. When the obstruction is repaired, the process suffers of a transitory increment in the tank temperature which can be seen in Figure 10.43 in $5.78 \times 10^5 \leq t \leq 6.00 \times 10^5$. This behavior is characterized by $Cluster5$ as can be seen in snapshot $t = 6.5 \times 10^5$. Hot water valve was stuck at 10% at $t = 6.50 \times 10^5$. This situation is not recognized by the system, because, it is not perceptible in the output signals. Figure 10.47 also shows the creation of $Cluster6$ in response to the new leak introduced in the process at $t = 9.00 \times 10^5$. This cluster can be seen in snapshots $t = 9.30 \times 10^5$ and $t = 10.10 \times 10^5$. It is worth remembering that only the three features with larger variance were selected as axes in the $3D$ plot, which means that even if some natural clusters overlap in the plotted dimensions they may be apart in the others (see as example $Cluster1$ and $Cluster6$ in snapshot $t = 10.10 \times 10^5$).

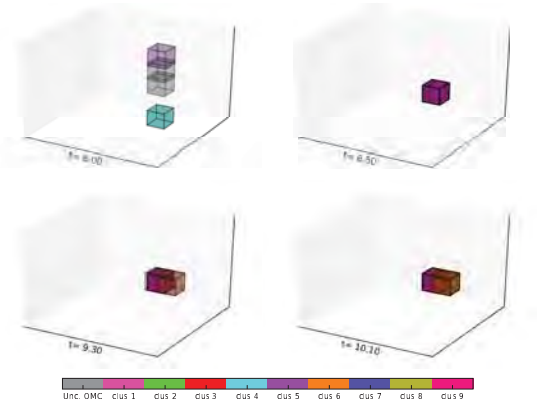


Figure 10.47: ***DyClee***'s μ-clusters evolution for Scenario 4 t between 6 and 10.10 seconds $\times 10^5$

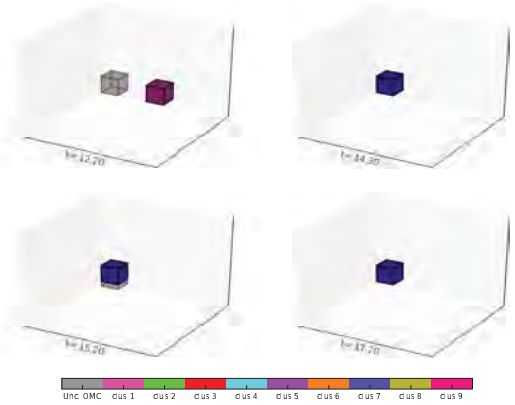


Figure 10.48: ***DyClee***'s μ-clusters evolution for Scenario 4 t between 12.2 and 17.2 seconds $\times 10^5$

Figure 10.48 starts with snapshot $t = 12.20 \times 10^5$ showing the change between $OP1$ and OP 2. Normal OP 2 behavior is characterized by $Cluster7$ as can be seen in the snapshots from $t = 14.3 \times 10^5$ onwards. At $t = 15.00 \times 10^5$ the steam valve is stuck at 10%: this behavior is represented by $Cluster8$ in snapshot $t = 15.20 \times 10^5$ (overlapping cluster). By the time $t = 17.2 \times 10^5$, only the normal behavior is latent in the system short time memory as can be seen in the last snapshot of Figure 10.48.

At $t = 18.00 \times 10^5$ a leak is introduced in the process simulation and ***DyClee*** tracks this new behavior which leads to $Cluster9$ (see snapshot $t = 18.3 \times 10^5$). Finally at $t = 20.00 \times 10^5$ the hot water valve is stuck at 40%, forcing the cold water input flow to increase in order to keep the tank level constant (process in closed-loop). The hot water valve blocks just at the same point as used in $OP1$, which leads the system to detect this behavior as $Cluster1$. This can be seen in snapshot $t = 20.50 \times 10^5$. Once this fault is repaired the system return to OP 2 as can be seen in snapshot $t = 22.20 \times 10^5$.

As a summary of the simulated scenario Figure 10.50 compares the clusters, i.e. operation modes, discovered by ***DyClee*** and the true operation modes. There is full concordance except for operation modes s_2 and s_4. The faulty state s_2 (hot water valve stuck at 10%) is indeed undiscriminable from operation mode $OP1$ since the hot water flow is not in use. The faulty mode s_4 is confused with

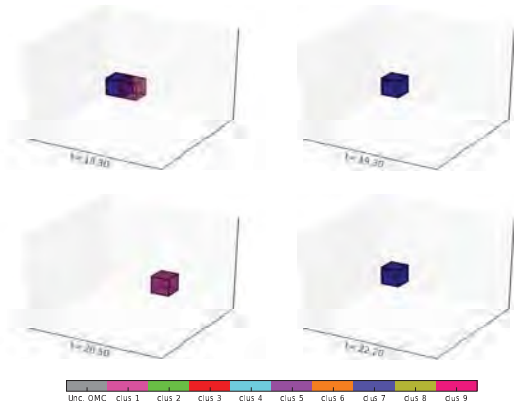


Figure 10.49: ***DyClee***'s μ-clusters evolution for Scenario 4 t between 18.23 and 22.2 seconds

$OP1$ which is not actually false because the observed behavior of the process corresponds exactly to operation mode $OP1$. Consequently, this mode is also undiscriminable using the available process data. The results produced by ***DyClee*** are hence very good.

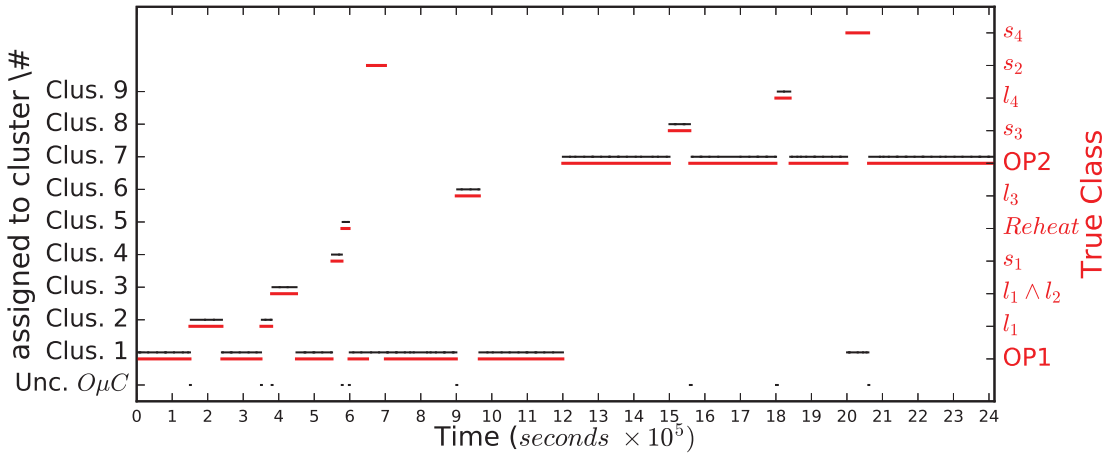


Figure 10.50: ***DyClee*** clustering results and CSTH Scenario 4 true labels

For supervision purposes, in order to make a proper diagnosis of the process, not only the measured variables but also the process set-points (control reference used as input to the closed-loop process) should be taken into account. We suggest to include the information about process set-points in a post-processing stage that would verify the relation of the current system measures with the desired system behavior.

Comparison with other streaming algorithms

DyClee results for Scenario 4 were compared against ClusTree [Kra *et al.* 11] and DenStream [Cao *et al.* 06] using four measures: the Silhouette index [Rou 87], the adjusted Rand index [Hub and Ara 85], Precision and Recall (as defined in Chapter 7). The best value for Precision and Recall is 1 and the

Algorithm	Silhouette	Rand	Precision	Recall
DyClee	0.7933	0.9302	0.9386	0.9386
ClusTree	0.9600	−2.4078	0.9991	0.5756
DenStream	1.0000	−0.0966	0.9968	0.7873

Table 10.6: Scenario 4 results comparison with other stream clustering algorithms

worst 0. For the silhouette coefficient, the best is 1 and the worst −1. The Rand index is ensured to have a value near 0 for random labeling, and exactly 1 for perfect labeling. The results provided in Table 10.6 show that the information found by ***DyClee*** is more representative than the information found by the other algorithms (bigger Recall). The Rand index indicates that ***DyClee*** results are the best compared to the true labeling.

10.2.2 From monitoring to modeling for decision-making support

In this subsection we show that ***DyClee*** learned timed DES allows it to distinguish behaviors that were non-diagnosables before (using only the dynamic clustering). Indeed, the DES allows to incorporate event based knowledge (for example coming from the control system) that is more adequately described with discrete dynamics. For complex processes the collection of set-points describing an operational point can be of an enormous size, comparable or even bigger than the observable measures used to observe the system behavior. These control variables change in a much slower time scale than the process dynamics. Giving its time scale and its potentially huge dimension, adding these control variables to the dynamic clustering directly increases its complexity in an unnecessarily manner. We propose instead to consider the desired operating point information as a discrete variable and include this variable directly in the DES.

As shown in Figure 10.50 ***DyClee*** can successfully track online the process and its evolution, however, the faulty state s_4 (Hot water valve stuck) is confused with $OP1$, since both have the same observable behavior. The key information that allow to distinguish between these two behaviors is the system desired OP. In industrial environments, the set of references that form an OP is sometimes equal or bigger in size that the set of output system measurements.

Considering the size of the set-points vector and under Hypothesis H9.2, we consider that adding all these variables to the clustering stage would increase its complexity and reduce its efficiency. Instead, we propose use the information of the system OP as a discrete variable representing the system discrete state (under Hypothesis H9.1).

As stated in chapter 9 in order to build a timed automaton from ***DyClee*** clustering results the continuous states (represented by the clusters) have to be paired with the discrete states. The association is made using the time as key. Table 10.7 show the pairs and the new labels assigned to those states that share the continuous state. Even if the DES is generated on-line at the same time as the clustering results, for the sake of brevity only the final results on the scenario are going to be depicted.

The transition table considers the new cluster labels found by the association of the dynamic clustering results with the available event information. The transition table columns and rows corresponds to the system states and each cell ij (with $i \neq j$) stores a number of transitions and an interval

Cluster	a	New label	Cluster	a	New label
1	$OP1$	**1**	5	$OP1$	5
	$OP2$	**1'**	6	$OP1$	6
2	$OP1$	2	7	$OP2$	7
3	$OP1$	3	8	$OP2$	8
4	$OP1$	4	9	$OP2$	9

Table 10.7: Final labels found pairing the continuous and discrete information

From		To cluster									
		Cl_1	Cl_2	Cl_3	Cl_4	Cl_5	Cl_6	Cl_7	Cl_8	Cl_9	$Cl_{1'}$
Cl_1	#	[97500, 297500]	2	0	1	0	1	1	0	0	0
	TI		[2500, 10000]	[-,-]	[20000]	[-,-]	[10000]	[40000]	[-,-]	[-,-]	[-,-]
Cl_2	#	1	[25000, 77500]	1	0	0	0	0	0	0	0
	TI	[0.0]		[10000]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]
Cl_3	#	1	0	[57500]	0	0	0	0	0	0	0
	TI	[0.0]	[-,-]		[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]
Cl_4	#	0	0	0	[5000]	1	0	0	0	0	0
	TI	[-,-]	[-,-]	[-,-]		[12500]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]
Cl_5	#	1	0	0	0	[5000]	0	0	0	0	0
	TI	[2500]	[-,-]	[-,-]	[-,-]		[-,-]	[-,-]	[-,-]	[-,-]	[-,-]
Cl_6	#	1	0	0	0	0	[52500]	0	0	0	0
	TI	[0.0]	[-,-]	[-,-]	[-,-]	[-,-]		[-,-]	[-,-]	[-,-]	[-,-]
Cl_7	#	0	0	0	0	0	0	[157500, 352500]	1	1	1
	TI	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]		[40000]	[10000]	[0.0]
Cl_8	#	0	0	0	0	0	0	1	[15000]	0	0
	TI	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[5000]		[-,-]	[-,-]
Cl_9	#	0	0	0	0	0	0	1	0	[27500]	0
	TI	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[0.0]	[-,-]		[-,-]
$Cl_{1'}$	#	0	0	0	0	0	0	1	0	0	[57500]
	TI	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[-,-]	[2500]	[-,-]	[-,-]	

Table 10.8: Transition table of the simulated scenario

$[t_1, t_2]$ characterizing the time constraints $cc_i \geq t_1$ and $cc_i \leq t_2$ (with $t_2 \geq t_1$). The diagonal of the matrix (cell ij with $i = j$) include the minimum and maximum $\mathcal{D}_i$ registered values. The transition table generated for the tested scenario is shown in Table 10.8.

To construct the DES the transition table rows i.e. the new labels are used as states. These states are represented as nodes in Figure 10.51. The edges correspond to the observable events, i.e. the departure and arrival from the natural clusters and the changes in the operational points. The initial state is determined as Cl_1 (the first encountered state). The transition functions correspond to all the non-empty cells in the transition matrix and are represented as edges in Figure 10.51. To determine the set of marked states the information about the number of operational points is used, so, in this case the are two accepted states. As stated in section 9 one clock is created for each state of the DES. Summarizing:

$$E: \{Clus_1, Clus_2, Clus_3, Clus_4, Clus_5, Clus_6, Clus_7, Clus_8, Clus_9, \overline{Clus_1}, \overline{Clus_2}, \overline{Clus_3}, \overline{Clus_4}, \overline{Clus_5}, \overline{Clus_6}, \overline{Clus_7}, \overline{Clus_8}, \overline{Clus_9}, OP1, OP2\}$$

$$\mathcal{S}: \{Cl_1, Cl_2, Cl_3, Cl_4, Cl_5, Cl_6, Cl_7, Cl_8, Cl_9, Cl_{1'}, \sim 1, \sim 2, \sim 3, \sim 4, \sim 5, \sim 6, \sim 7, \sim 8, \sim 9, \sim 1'\}$$

$$\mathcal{S}_0: \{Cl_1\}$$

$$
\begin{aligned}
& < Cl_1, \quad (\overline{Clus_1}), cc_1, && > \rightarrow \sim 1 \\
& < \sim 1, (Clus_7, OP2), \quad , cc_1 \in [40000, 40000] > \rightarrow Cl_7 \\
Tf_t: \quad & < \sim 1, (Clus_2, OP1), \quad , \ cc_1 \in [2500, 10000] > \rightarrow Cl_2 \\
& < \sim 7, (Clus_1, OP2), \quad , && > \rightarrow Cl_{1'} \\
& \ldots
\end{aligned}
$$

$\mathcal{S}_m$: $\{Clus_1, Clus_7\}$

$\mathcal{CK}$: $\{cc_1, cc_2, cc_3, cc_4, cc_5, cc_6, cc_7, cc_8, cc_9, cc_{1'}\}$

Figure 10.51 shows the timed automaton diagram at the end of the simulated Scenario. This diagram, generated from ***DyClee***'s results, depicts the 10 clusters and the corresponding transitions. States Cl_1 and Cl_2 are found to be the marked states (double-circle representation). The only events depicted explicitly are the OP, nevertheless each transition from a Cl_i state to a $\sim i$ state implies the $\overline{Clus_i}$ event and each transition from a $\sim i$ state to a Cl_j state implies a $Clus_j$ event. Each cluster state includes, below the cluster identification, the information related to the median time that the system had remained in each of these states. Clocks constraints (in Cl_j to $\sim i$ transitions) and resets (in Cl_j to $\sim i$ transitions) are included in the graph according to the notation explained before. The probability of a transition is represented as $P_{i \rightarrow j}$ at the beginning of each departing transition for the $\sim i$ states. For the Cl_i states the only allowed transition is to $\sim i$, fired by the $\overline{Clus_i}$ event. Since this transition has a 100% probability of occurrence it is not depicted in the graph. A red colored font indicates the current state. The orange polygon form depict the nodes generated by the inclusion of the discrete dynamics. In these nodes the information of the continuous twin state is also depicted as reference, that is the label found by the clustering process.

It can be seen that the state Cl_1 ($OP1$) is the focus of the diagram having transitions to states Cl_2, Cl_4, Cl_6 and Cl_7. The diagram also shows that the state Cl_5 is only achievable from Cl_4 and that Cl_3 only from Cl_2. These relations show the strong dependency of these states to their sources. From a point of view of supervision these transitions indicate that the faulty states Cl_3 and Cl_5 are a degradation of already degraded states. The system current state at the end of the simulated scenario is Cl_7, meaning that the system is currently in $OP2$ and no faulty behavior is currently detected. The state $Cl_{1'}$ that has the same continuous behavior that Cl_1 is now diagnosable since its only achievable from Cl_7 when the system is in $OP2$. As a summary of ***DyClee*** results from clustering and discrete modeling, Figure 10.52 compares the clusters discovered by ***DyClee*** and the true operation modes.

10.3 Summary

In this chapter ***DyClee*** supervision capabilities were tested using industrial-like benchmarks. Section 10.1 illustrates ***DyClee*** performance in systems with varying operational points following periodical cycles. Section 10.2 showed ***DyClee*** performance in a dynamic system with slower dynamics in which the drifts and shifts in data were mostly related to faults.

It was shown that ***DyClee*** dynamic clustering is useful to monitoring and tracking dynamic systems in their operational modes even when their behaviors drift in time responding to changing environments (subsection 10.1.1). ***DyClee*** has also proved to be capable to detect different types of

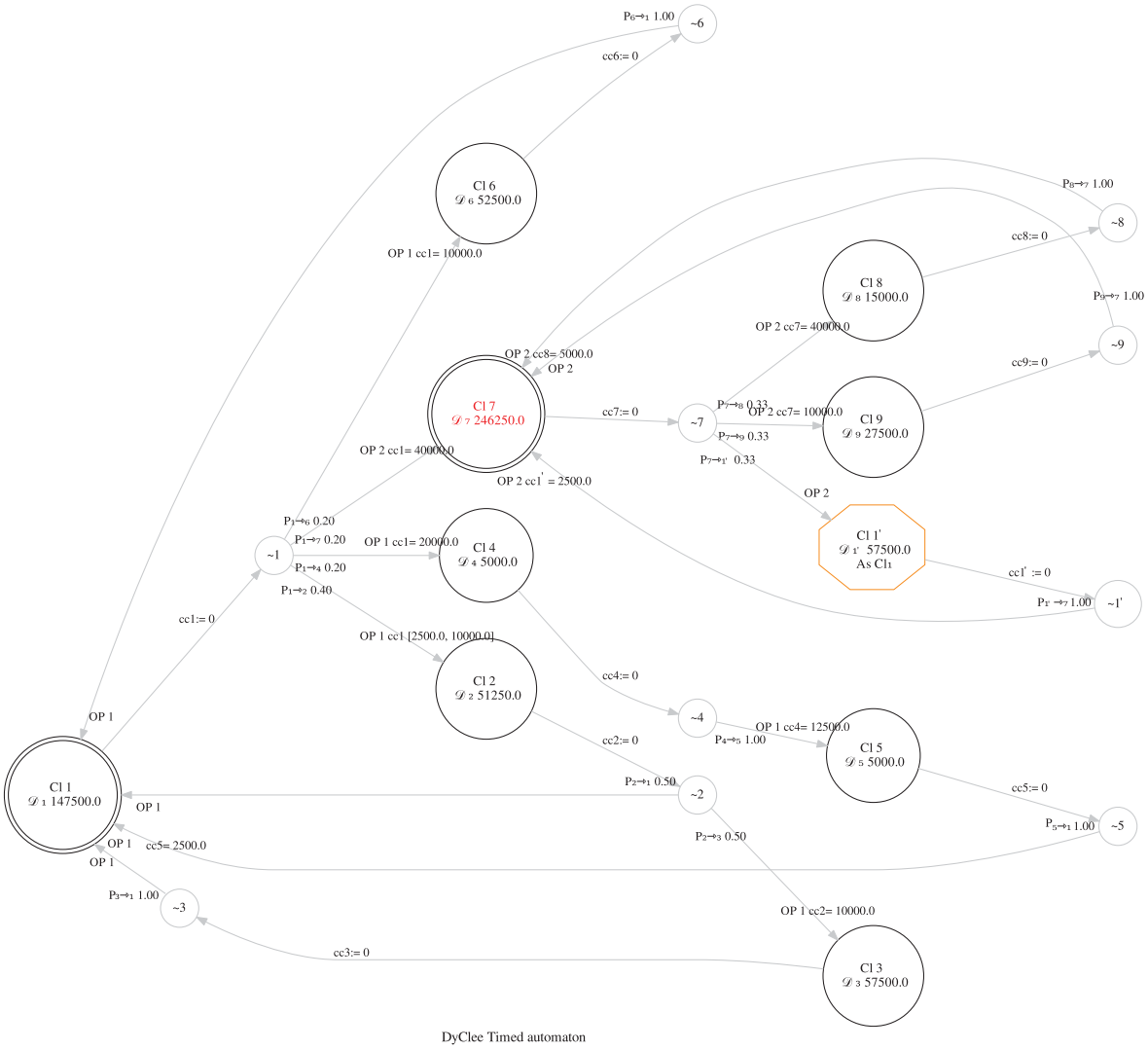


Figure 10.51: ***DyClee*** timed automaton generated from clustering results

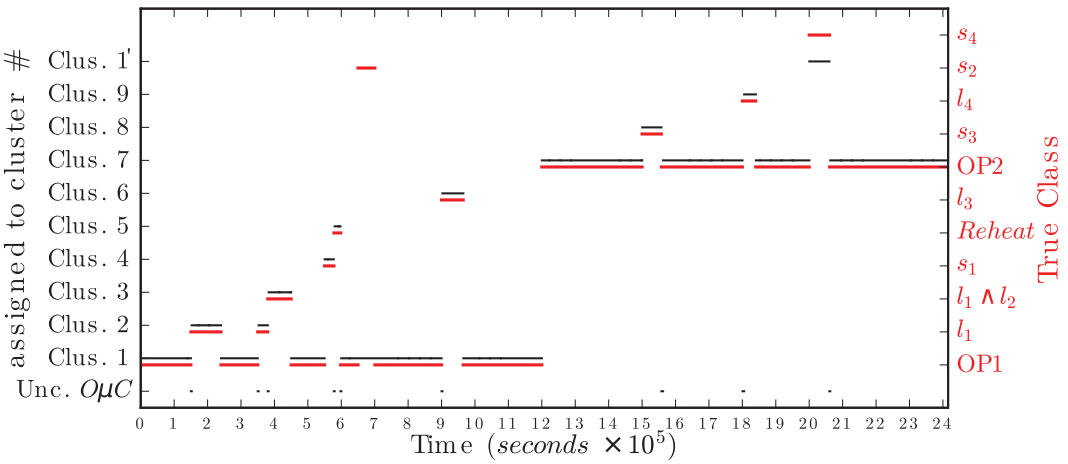


Figure 10.52: Final diagnosis achieved by coupling continuous and discrete information

faults including those with time varying magnitudes and those that are non persistent on a continuous stirred tank heater benchmark. These faults might occur single or multiple (subsection 10.2.1).

The tested processes allowed to prove that dynamic clustering results can be used to automatically generate a DES model of the system. ***DyClee*** DES generator is even capable of detecting the marked states having as input only the number of marked states that should be supplied by the process expert.

Conclusions and perspectives

Conclusions

The research area of dynamic clustering is new and challenging from both the theoretical and the practical point of view. There are a huge number of applications in which considering the temporal evolution of the system or the objects passing through is necessary to perform a proper monitoring. Several of these applications capture huge amounts of process information in stream and have limited possibilities to process and store all the data. So far only a limited number of algorithms for dynamic clustering of dynamic systems under evolving environments have been developed. These algorithms can be separated into two categories: methods classifying dynamic objects with dynamic parameter classifiers and methods classifying dynamic objects with dynamic structure classifiers. The first group is formed by algorithms with a static structure generally found during a training stage and fixed for its application in data recognition. In these algorithms only the cluster parameters can be updated in the recognition stage for which most of these algorithms are able to detect only gradual changes in the data structure. In fact, they frequently require a re-learning process when new nominal operation modes or new components are added to the process. The second group is formed by algorithms that can update both their structure and their parameters in response to detected changes in the system data. Most of the algorithms in this group classify data using adaptive incremental learning and, in cases where data arrive in high-volumes, using a streaming approach. To achieve computational and real-time limitations several dynamic techniques process dynamic data without taking into account temporal information.

This thesis has suggested a novel algorithm for dynamic clustering which can be applied to any process in which a sufficient amount of data, describing the process behavior, is available. This **Dy**namic **Cl**ustering algorithm for tracking **E**volving **E**nvironments (***DyClee***) surpasses previous approaches due to its ability to learn and recognize system behaviors on-line, rejecting outliers and adapting automatically the clusterer structure and parameters to follow changes in system data. ***DyClee*** makes use of the advantages of distance-based clustering along with density-based clustering to handle non-linear, multi-density distributions even in high overlapping situations. The proposed algorithm comprises four modular algorithms than can work together (in parallel) for monitoring evolving processes in real-time. Each of these algorithms was conceived to help in the monitoring, even in cases where there is no knowledge about the process. In specific, the main contribution of this thesis is the proposed algorithm for dynamic clustering of static and dynamic objects presented in Part II.

The first module, presented in Chapter 5, groups similar samples into a summarized statistical representation called μ-cluster according to an L_1 distance similarity measure. The choice of the distance was done based on its meaningfulness in medium to high dimensional spaces. μ-clusters are updated using the information of the samples assigned to them and weighted by a forgetting coefficient. Several functions were proposed as forgetting functions, giving the algorithm the capacity to adapt to different dynamics acting at different time scales. The forgetting process affects the μ-clusters making them prone to mimic current system dynamics quicker than non dynamic clustering approaches. The use of two lists to store active and less-active μ-clusters gives priority to the evolution of known behaviors and speeds up the sample location process. The implemented algorithm takes process data directly from text or log files and/or databases and processes it online in an incremental fashion. This

incremental approach diminishes memory requirements, allowing to process very large data sets, and even data measurements arriving in a high speed stream directly from the process.

The second module, presented in Chapter 6, reads the μ-clusters information provided by the first module and analyses both μ-cluster proximity and density. The principal assumption in this algorithm is that dense μ-clusters that are close enough (connected) belong to the same final cluster. To diminish the computational load of checking for μ-cluster groups inside the μ-cluster set a tree indexing structure was implemented (cf. Appendix A). Two different approaches to establish the dense character of a μ-cluster, named global-density approach and local-density approach were proposed. μ-clusters are characterized as belonging to one of three categories: dense μ-clusters ($\mathbb{D}\mu$C), semi-dense μ-clusters ($\mathbb{S}\mu$C) and low density outlier μ-clusters ($\mathbb{O}\mu$C). In the global-density analysis outliers correspond mainly to deviated data but transitional or infrequent states can also be characterized as $\mathbb{O}\mu$C. This analysis allows us to detect clusters with similar densities while the local-density analysis allows the detection of clusters with varied densities. In fact, using the local-density analysis, low density populations (as faults) can be represented as well as high density populations (as is usually the case of normal behavior). In addition, the local-density analysis allows the detection of novelty behavior in its early stages when only a few samples giving evidence of this evolution are present.

The fact that ***DyClee*** natural clusters are formed as groups of smaller units following dynamic data (μ-clusters), gives the algorithm the capacities of tracking cluster's evolution and automatically merge and split clusters without loosing information and without needing all the belonging samples to be stored. New clusters are created as soon as a group of μ-clusters reach a relative high density (with respect to global or local thresholds) and old non-representative clusters are removed to keep up an updated cluster distribution. In ***DyClee*** the distance-based and density-based clustering algorithms are implemented as separate modules that run in parallel, i. e. using independent memory spaces and CPU resources. This implementations make possible to operate even in distant computers provided that a bidirectional pipe is available.

A second contribution refers to dynamic feature extraction. The algorithm presented in Chapter 8 allows ***DyClee*** to process time depending features extracting the time related information into *episodes* that can then be used as ***DyClee***inputs. Episodes allow ***DyClee*** to capture the information about the evolution of the monitored characteristics. The episode abstraction is done for each monitored characteristic using polynomial approximation over non overlapping sliding windows and then packed into epochs. Epochs carry information not only about the shape and magnitude of the represented behavior but also about its time of occurrence. The polynomial coefficients found for the time series section are optimal with respect to the denoised signal approximation found using the cosine transform (cf. Chapter 8). The third module of ***DyClee*** implements the feature extraction mechanism explained before. This module is optional in ***DyClee*** hence the user can choose whether or not to activate it.

A third contribution is the automatic generation of a timed automaton from ***DyClee*** clustering results presented in Chapter 9. This automaton improves clustering results interpretability for decision-making purposes and improves the fault detection capabilities by the inclusion of event related dynamics. This adaptive model, built automatically by the fourth module as new data of the system is gathered, provides a high level abstraction of the system with information about the system past and current states along with the possible transitions identified by the time of transition

and the time constraints between two states.The timed automaton graphical representation allows the operator to see rapidly an overall description of the system including the current state that is identified (red color). Information about the merge of clusters is also included in the diagram inside each state in blue color. In the cases were information about the number of accepted nominal states (marked stated) is available, ***DyClee*** DES generator makes an estimation of which of the states form the subset of marked states, that is, the set of states in which the system is allowed to remain.

DyClee clustering results include a cluster general description that along with ***DyClee*** timed automaton describe the active system dynamics. This cluster description includes, among other things, the feature range and center of gravity which can be used to generate a radar chart. This kind of graphical representation reflects in two dimensions vectors of three or more dimensions providing a graphical signature of the system states.

Chapter 10 presents the application of ***DyClee*** for the monitoring of two different benchmarks, a steam generator part of a pilot thermal power plant and a continuous stirred tank heater, a common reactor used in chemical engineering. The first use case, that of the steam generator, allows to show ***DyClee*** automatic reconfiguration capabilities and its robustness to the order in which samples are presented to it. In this example the system under analysis exhibits highly non linear cyclic dynamics and it was shown that ***DyClee*** can successfully track all the system's operational modes. ***DyClee*** results are consistent with the information provided by the system expert. The second use case showed that ***DyClee*** dynamic clustering is useful to monitor and track dynamic systems in their operational modes even when their behaviors drift in time responding to changing environments. With this use case ***DyClee*** has also proved to be capable to detect different types of incipient and abrupt non persistent faults of varying magnitudes. The ability to differentiate single faults from multiple faults was also shown.

The tested processes allowed to prove that dynamic clustering results can be used to automatically generate a DES model of the system including timed constraints and structural changes. In order to provide help for decision making, additionally to the automaton diagram, a radar chart and trend charts are generated in ***DyClee*** implementation. All the graphical results shown in Chapter 10 were generated directly from the Python implementation of ***DyClee***.

In this thesis a framework for the characterization and monitoring of complex systems based only on measurements were presented. The proposed algorithms were conceived for supervision purposes but their applications to other fields is completely open. The next section presents possible research directions and applications for this work.

Possible research directions

As stated above the developed clustering algorithm was tested in pilot plants of industrial processes, nevertheless the used methodology is general so its application in other fields of knowledge, in which large amounts of data are available, is promising. Possible applications include intrusion detection in networks [Maz *et al.* 11], tracking of moving objects in images or video [Jim *et al.* 11], event detection and chronicle recognition for DES [Sub *et al.* 14], auto-compliance evaluation for in-circuit-test equipment [Jay and Muh 14], etc.

The developed algorithm implementation lacks a graphical user interface (GUI) hence is accessible only by terminal commands. A personal goal as future work is to implement a GUI that allows the use of the developed software not only for research purposes but also for educational purposes. For example, the proposed algorithm could be used for the monitoring of remote academic laboratories [Bar *et al.* 13] foster the inclusion of supervision concepts in engineering education.

Regarding possible improvements to the dynamic clustering algorithm, Monrousseau *et. al* have provided some evidence that the use of fuzzy intervals instead of crisp intervals can improve classification results even when the signal to noise ratio is high [Mon *et al.* 15]. Based on these results it seems interesting to implement fuzzy μ-clusters that could potentially improve the classification of highly noisy time series.

An aspect that was not addressed in this work was the feature selection problem. The current implementation of ***DyClee*** allow feature selection in two cases. If a feature's operational range is found to be smaller than the μ-cluster relative size, it is not possible for the algorithm to distinguish changes in this variable and thus its processing is unnecessary. The second possibility is to remove those features for which the feature range is exactly the same for all the recognized clusters. Nevertheless, it should be noticed that if continuous learning is desired all features should be kept, since features that were negligible in the past can as well be the key for characterizing a novel behavior. An example of this was presented in Section 10.2 with the CSTH use case. In this example, for Scenario 1 and Scenario 2 the tank level and temperature remain constant for the whole simulation, then for the leak recognition in Scenario 3 the tank level was the key feature. Scenario 4 also showed that the steam valve blocking fault was detectable only in the tank temperature, a feature that is negligible for the other behaviors.

In the era of big data multiple industries look for algorithms that can handle large amount of data in streams, designed to make use of clusters of computers under a distributed processing framework as Hadoop[3]. ***DyClee*** was designed to run each of its stages independently and in parallel using one or several computers. For the moment the parallelization is implemented at algorithm level but it would be interesting to paralellize intra-stage calculations such as the search for connections between μ-clusters. To achieve that level of parallelism an approach based in distributed input data should be considered. The python implementation developed in this thesis can be used as starting point to develop an algorithm compatible with distributed file systems (for process historical data) and with stream oriented message brokers (for current system measurements) as Apache Kafka [Apa 16b].

Regarding the developed DES generator multiple aspects can be source of further research. As stated in chapter 9, under hypothesis H9.1 and H9.2 faults that can be detected thanks to the DES model correspond to one of three types: 1) occurrence of an event labeling a transition not followed by the actual transition, 2) state transition without the occurrence of the event labeling the transition, 3) state transition that does not correspond to the event that has occurred. The detection of the type of faults is achieved by the verification of the DES generated model. In this thesis the DES model generation was automated but the verification was left for the system operator. There are no doubts about the improvement that an automatic verification module can provide for supporting

[3] "The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage" [Apa 16a]

decision-making supervision. Actually, the generation of a discrete event model opens the door to the possibility of modeling and monitoring hybrid systems, a growing research field in which great challenges are still open.

Another improvement can come from the prognosis point of view. In [Ram and Gou 14] the use of one-class classifiers to find the remaining useful life of a system was evidenced. Arguably the dynamic structure of ***DyClee*** can improve prognosis by the capability to learn possible modes in an automatic fashion, this path should be explored in future works. ***DyClee*** implementation allows the user to see all possible (already encountered in the past) transitions departing from the system's current state and assigns a probability to these departing transition, based on their past occurrences. At the moment, key information that might potentially improve the transition probability estimation such as the state density or the already characterized timed constraints are not taken into account. The use of the transition statistics along with states density values could potentially give a measure about the possibility of firing a transition over another given a temporal constraint. An alert to the system operator should be considered if the possibility of transitioning to a failure state increases.

Appendix A

Comparison of Spatial Indexes

An spatial index was set-up to speed-up μ-cluster connection search. Three different tree data structures were tested in order to find the index structure best suited to our goals. The selected structures are: the R*Tree [Bec *et al.* 90], the BallTree [Omo 89] and the KDTree [Ben 75].

The R*Tree, or rectangular Tree, was proposed in 1990 and has been largely used since. In clustering applications, the ClusTree algorithm uses an extension of this index to efficiently locate the right place for object insertion [Kra *et al.* 11]. The key idea of this data structure is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree. BallTrees is a completely binary tree in which each node refer to a region bounded by an d-dimensional hyper-sphere [Omo 89]. A parent node is hence the smallest hyper-sphere that contains all the hyper-spheres of its children. The BallTree is the only index, among the analyzed indexes, that accepts the intersection of sibling regions. A K-dimensional binary search tree (KDTree) partitions the data space into mutually exclusive hyper-rectangular regions. Each region is found by recursively splitting the space using axis-parallel hyperplanes. The splitting process finishes when each sub-region has a number of points less than or equal to a given threshold.

The python implementations of the R*Tree[1], the BallTree[2], the KDTree[3] and the cKDTree[4] (A cython implementation of the KDTree), were tested using two different scenarios. The computation time and memory consumption (maximum resident set size used), were selected as comparative measures.

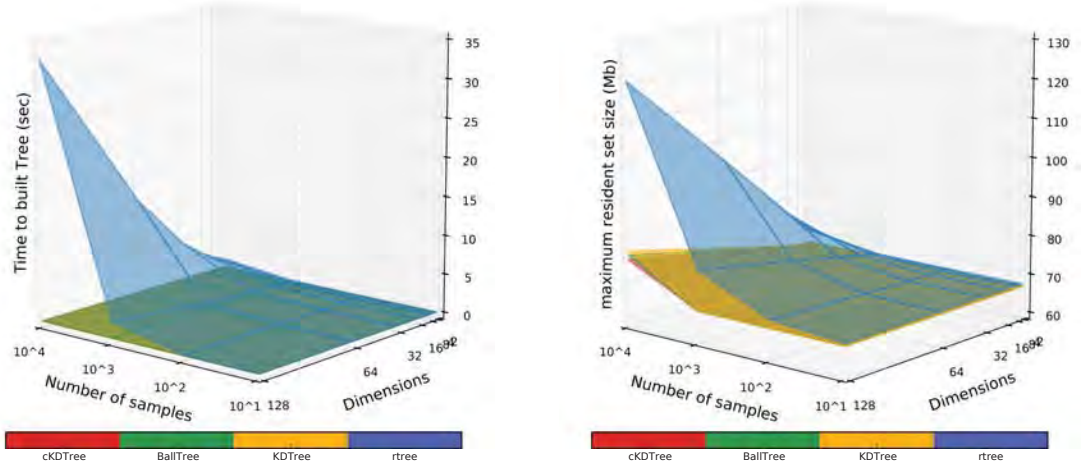
As stated before the index should increase the efficiency of the closest μ-cluster retrieval in the first stage of ***DyClee*** and also the group retrieval in the second stage. Group retrieval is considered as the process of finding all the neighborhoods in the data set. This process involves a recursive implementation of radius-neighbor queries. Time and memory measures were taken for the tree construction process and for the group retrieval process. These measures are the average value over a hundred simulations.

In the first scenario, data correspond to random samples extracted from a uniform distribution. The algorithms were tested for a data sets containing n data samples, where each sample is a d-dimensional vector. Tests were performed for n varying in the range $n \in \{10^1, 10^2, 10^3, 10^4\}$ and d in $d \in \{2^1, 2^2, 2^3, 2^4, 2^5\}$.

[1] http://toblerity.org/rtree/
[2] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html
[3] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html
[4] http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html

Figure A.1 shows the resources used for the tree building task. Computation time measured in seconds is presented in Figure A.1(a) and the maximum resident set size measured in Mb is shown in Figure A.1(b). It can be seen that the R*Tree implementation performs poorly with respect to the other indexes. The actual measurements of the computation time and maximum resident set size are presented in tables A.1 and A.2 respectively.

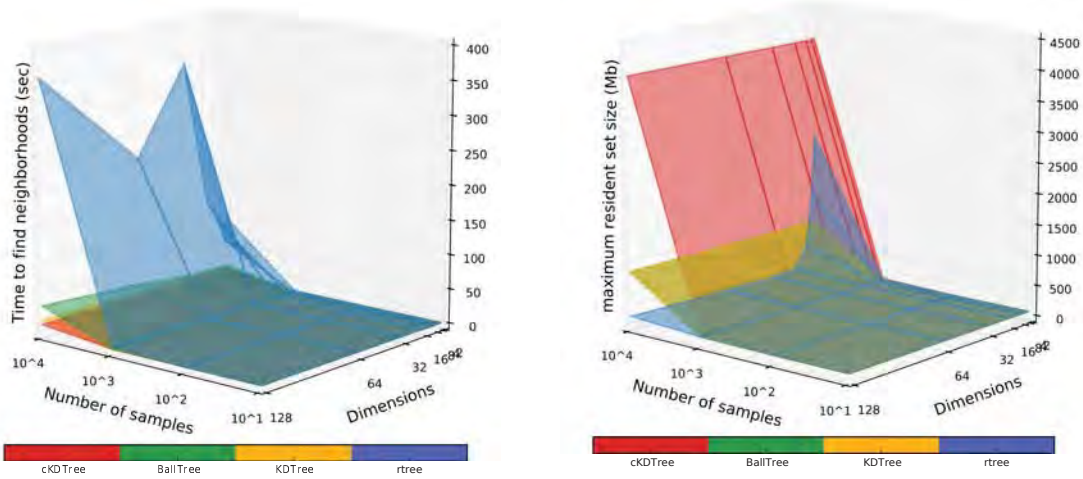
The group retrieval task is computationally more expensive as can be seen in Figure A.2, where the computation time for the worst case (R*Tree, $n = 18$,$d = 10^4$) passes from 33.12 seconds in the tree building task to 362.9 seconds for group retrieval (see Figure A.2(a)). The best computation time is achieved by the cKDTree index but at cost of having the worst memory consumption (see Figure A.2(b)). The recorded data for the group retrieval task is presented in tables A.3 and A.4.



(a) Computation time

(b) Maximum resident set size

Figure A.1: Comparison between different tree indexes for the tree building task in a uniform distribution



(a) Computation time

(b) Maximum resident set size

Figure A.2: Comparison between different tree indexes for the group retrieval task in a uniform distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	12.12	13.62	14.50	16.12	17.37	23.25	31.62
	2	94.88	99.62	108.62	128.00	160.00	222.75	358.50
	3	1142.25	1236.62	1444.00	1977.12	3456.00	8483.38	26073.62
	4	13861	15552	19784	30752	60609	153557	414038
BallTree	1	1.75	1.87	1.87	1.87	1.87	1.87	1.87
	2	1.87	2.00	2.00	2.12	2.38	2.62	3.25
	3	4.00	4.25	5.25	7.50	11.87	20.37	36.37
	4	40.25	47.50	66.12	103.12	203.75	424.12	877.87
KDTree	1	1.87	1.87	1.75	1.75	1.87	1.87	1.75
	2	1.87	1.87	1.87	2.00	2.12	2.38	3.00
	3	3.50	3.75	4.50	6.00	9.00	16.62	31.62
	4	33.62	40.25	54.25	84.50	165.50	364.37	786.75
cKDTree	1	1.00	1.00	1.00	1.12	1.00	1.12	1.12
	2	1.12	1.12	1.12	1.25	1.25	1.50	1.75
	3	2.25	2.38	2.62	3.12	3.87	5.00	7.12
	4	15.12	16.38	18.75	23.88	33.75	46.25	67.88

Table A.1: Tree building computation time (relative to 80μsec) for uniformly distributed data

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.01	1.01	1.02	1.03	1.05	1.09	1.16
	4	1.05	1.06	1.08	1.13	1.23	1.43	1.81
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.01	1.03	1.07	1.15
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.01	1.03	1.08	1.17
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.01	1.03	1.06	1.14

Table A.2: Tree building maximum resident set size (relative to 66.762Mb) for uniformly distributed data

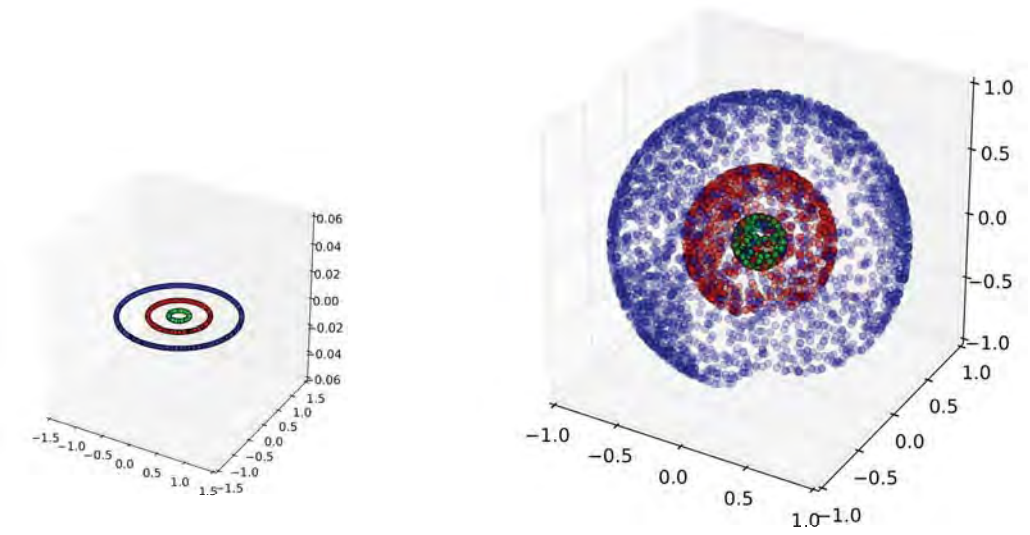
Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	4.00	5.23	5.17	5.83	7.29	11.63	17.14
	2	49.83	48.77	62.66	86.66	85.66	124.57	207.80
	3	2339	1843.80	1988.77	3330.66	3424.17	6092.80	11293.66
	4	260650	171165	254612	350371	967009.14	614249	1036927
BallTree	1	1.06	1.06	1.06	1.06	1.09	1.09	1.14
	2	6.43	6.54	6.63	7.11	7.71	9.09	11.26
	3	511.94	510.60	520.17	541.06	603.00	736.14	953.86
	4	50449	49619	50422	51687	58331	81430	109406
KDTree	1	1.06	1.06	1.06	1.06	1.06	1.09	1.09
	2	6.46	6.43	6.40	6.57	6.57	6.71	7.00
	3	514.34	514.26	516.29	515.03	509.11	512.94	518.91
	4	49768	50382	50643	49987	50149	49846	49907
cKDTree	1	1.03	1.03	1.00	1.06	1.03	1.09	1.11
	2	6.14	6.17	6.14	6.14	6.29	6.54	6.89
	3	331.94	333.94	335.34	335.94	334.77	336.54	348.29
	4	30971	31098	31191	31205	31172	31185	31249

Table A.3: Group retrieval computation time (relative to 350μsec) for uniformly distributed data

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.32	1.20	1.09	1.05	1.06	1.11	1.19
	4	34.63	20.14	7.27	1.88	1.36	1.58	2.07
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.74	12.74	12.74	12.76	12.80	12.88	13.04
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.11	1.12	1.14
	4	12.74	12.74	12.74	12.76	12.81	12.89	13.05
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.47	1.47	1.47	1.47	1.48	1.49	1.51
	4	59.66	59.67	59.68	59.70	59.76	59.87	60.10

Table A.4: Group retrieval maximum resident set size (relative to 66.762Mb) for uniformly distributed data

The second scenario measures the indexes performance (computation time and maximum resident set size) for the tree building and group retrieval tasks in a spherical distribution. The 2-dimensional version of this distribution, formed by concentric circles, is shown in Figure A.3(a). In higher dimensions this distribution represents samples located in the surface of concentric hyper-spheres. For illustrative purposes a 3-dimensional spherical distribution is depicted in Figure A.3(b).



(a) Spherical distribution in 2D used to test

(b) Spherical distribution in 3D (illustrative)

Figure A.3: Representation of the spherical distribution for 2 and 3 dimensions

Performance statistics for the tree building task are shown in tables A.5 and A.6 and its graphical representation depicted in Figure A.4. In general all the indexes perform better in the tree building task for this distribution that for the uniform distribution as can be seen in Figure A.5. The group retrieval measurements are shown in tables A.7 and A.8.

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	15.50	16.00	16.83	18.50	21.83	28.50	42.50
	2	127.83	134.00	147.83	168.83	216.33	298.83	487.17
	3	1524.67	1640.83	1912.83	2613.17	4548.17	10029.67	27845.00
	4	18678	21197	26249	36314	59633	1302993	374534
BallTree	1	2.17	2.17	2.17	2.17	2.17	2.17	2.17
	2	2.33	2.33	2.33	2.50	2.83	3.33	4.17
	3	5.17	5.50	6.83	9.67	15.67	26.67	49.50
	4	56.00	62.67	90.00	139.00	275.17	576.17	1180.50
KDTree	1	2.17	2.17	2.17	2.17	2.17	2.17	2.17
	2	2.33	2.33	2.33	2.50	2.67	3.00	3.83
	3	4.50	4.83	5.83	7.67	11.83	22.33	43.33
	4	45.67	52.50	72.83	113.50	216.33	492.67	1060.67
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.17	1.17
	2	1.17	1.17	1.17	1.33	1.33	1.67	2.00
	3	2.67	3.00	3.00	3.83	5.00	6.33	9.33
	4	19.67	21.83	31.00	31.83	44.67	62.50	91.67

Table A.5: Tree building computation time (relative to 60μsec) for spherical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.01	1.02	1.02	1.03	1.05	1.09	1.16
	4	1.05	1.06	1.08	1.13	1.23	1.42	1.80
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44

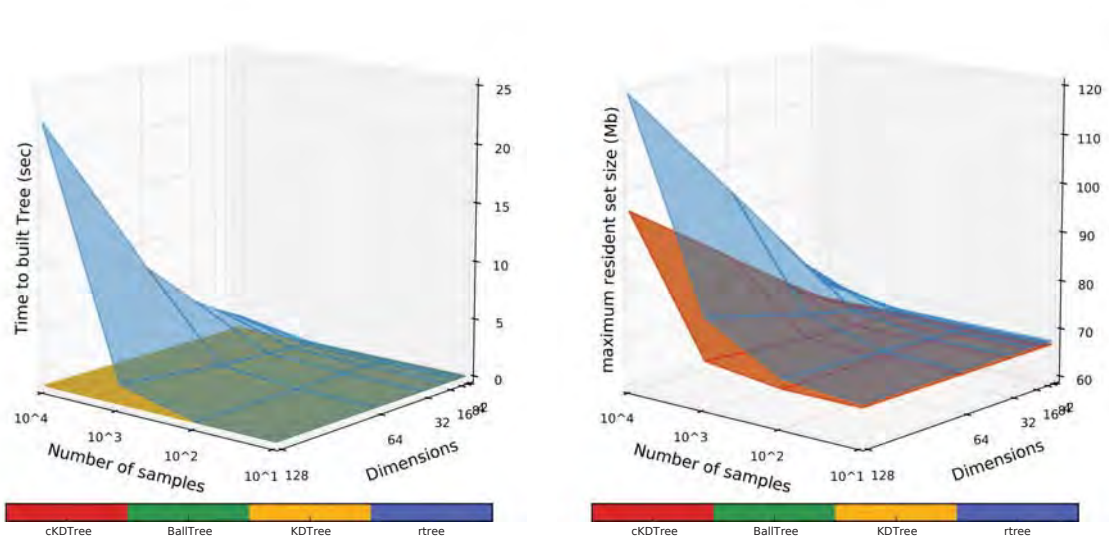
Table A.6: Tree building maximum resident set size (relative to 66.64Mb) for spherical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	3.55	3.80	3.80	4.58	6.07	9.10	15.10
	2	32.60	40.12	49.67	65.12	90.73	125.00	191.50
	3	1920.95	1955.17	2094.47	2451.57	3199.20	4710.50	7680.00
	4	219311	236073	267983	317122	402437	528211	804337
BallTree	1	1.05	1.15	1.20	1.28	1.60	2.17	3.00
	2	5.75	5.60	5.62	5.77	5.72	5.85	5.95
	3	458.25	450.72	455.75	450.77	456.17	451.57	463.32
	4	44230	44562	44349	44420	44831.72	44533	44381
KDTree	1	1.00	1.15	1.17	1.28	1.50	2.25	3.00
	2	5.58	5.77	5.62	5.70	5.70	5.88	6.10
	3	448.47	454.30	459.42	456.32	459.35	455.32	457.50
	4	44812	44817	44637	44389	44525	44722	44339
cKDTree	1	1.10	1.20	1.30	1.38	1.60	1.92	2.45
	2	5.47	5.53	5.45	5.45	5.65	5.67	6.00
	3	300.75	298.70	301.20	301.38	311.93	299.07	301.97
	4	27416	27553	28353	27688	27607	27688	27730

Table A.7: Group retrieval computation time (relative to 400μsec) for spherical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.53	1.53	1.54	1.55	1.57	1.62	1.72
	4	59.97	59.99	59.97	60.02	60.87	63.45	64.32
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.76	12.76	12.77	12.78	12.83	12.90	13.06
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.76	12.76	12.77	12.79	12.83	12.91	13.08
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.47	1.47	1.47	1.47	1.48	1.49	1.51
	4	59.77	59.78	59.79	59.81	59.87	59.98	60.21

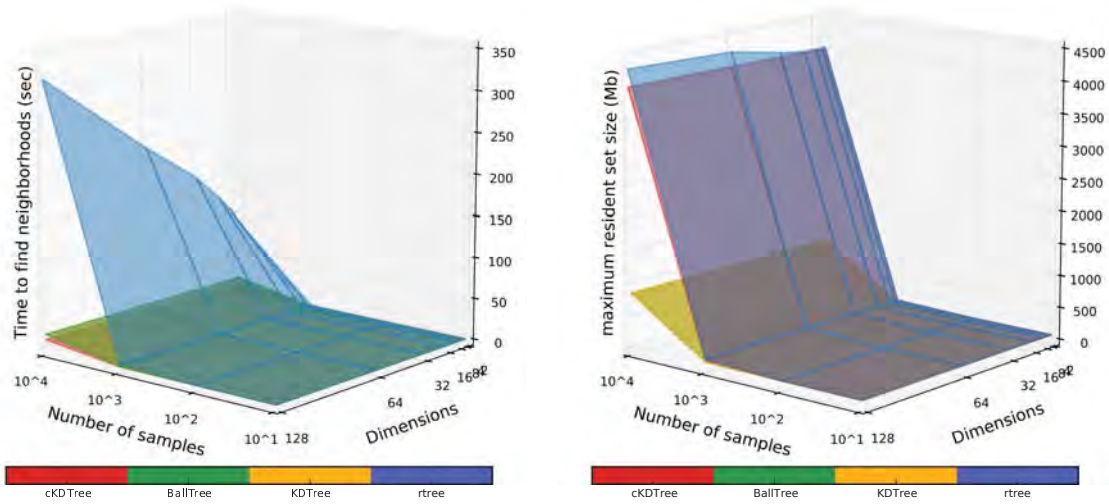
Table A.8: Group retrieval maximum resident set size (relative to 66.64Mb) for spherical distribution



(a) Computation time

(b) Maximum resident set size

Figure A.4: Comparison between different tree indexes for the tree building task in a spherical distribution



(a) Computation time

(b) Maximum resident set size

Figure A.5: Comparison between different tree indexes for the group retrieval task in a spherical distribution

Appendix B

Possible and required parameters of *DyClee*

In the previous chapters the ***DyClee*** algorithm was introduced and its stages explained. This chapter focuses on ***DyClee*** implementation. The developed dynamic clustering algorithm was implemented in Python 3. Python is an interpreted, interactive, object-oriented programming language that perfectly matches power with readability of the written code. This open-source language runs in all the platforms and can be easily shared with other scientist.

The implementation of ***DyClee*** allow users to analyze and cluster data sets using only two lines of code. The first line will establish ***DyClee*** configuration and the second will run the clustering algorithm in the provided data. Data can come from a database, a text file or can be sent to the algorithm in the run instruction.

The configuration of ***DyClee*** is perform via key parameters. ***DyClee*** offers several optional features that might help the user improving clustering by adding some knowledge about the data. Information about the distribution of data, for example, there are clusters with densities in different levels, or, each clusters contains approximately the same amount of elements; could improve data clustering if known, and even if there are unknown user might hypothesize about it for data analyses purposes. Knowledge about distributions changing in time can also be taken into account, giving for example, more importance to recent data.

DyClee optional and required features are explored in this chapter. For each parameter its name and a brief description is given. When it is considered as necessary, an example of use is also provided.

B.0.1 Hyper-cube size

Parameter name: `portion`
Input type: float value
Default value:
Description: Box size per dimension.

As stated in Chapter 4 a μ-cluster μC_z is a d-dimensional hyper-cube with of size $S_z^j \ \forall j$. Since features are normalized, the feature range are $[0, 1]$. The `portion` describing the size of the box in each dimension could be a float number in the interval $(0, 1)$, even if a value of `portion`> 0.5 may not make sense. This parameter is **the only required** parameter of ***DyClee***.

B.0.2 Uncommon dimensions

Parameter name: `uncdim`
Input type: Integer value
Default value: Zero
Description: Selectivity of the distance-based clustering stage.

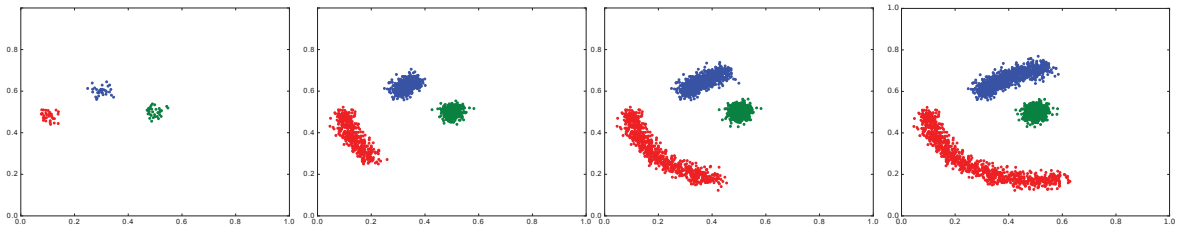
As stated in Chapter 6 two clusters are said to be *directly connected* if their hyperboxes overlap in all but φ dimensions. The `uncdim` option establishes the φ parameter, setting the feature selectivity of the clusterer. By default `uncdim=0`, thus direct connection is established when hyperboxes overlap in all dimensions ($\varphi = 0$). This parameter may be particularly helpful when analyzing data sets for which little or none knowledge is available in medium to high dimensional spaces.

B.0.3 Forgetting function

Parameter name: `forget_method`
Input type: String input.
Default value: `"None"`
Description: Mathematical function used to emulate the forgetting process. Accepted strings are: `"linear"`, `"expo"`, `"sigmo"`, `"z_funct"`, `"trapezoidal"` and `"half_life"`.

As stated in subsection 5.2.1 ***DyClee*** implements a forgetting process in order to modify the knowledge representation, hence achieving a correct evolution tracking of the process. Users can chose weather to use or not the forgetting functions in their models. By default this option is deactivated meaning that no forgetting function is applied to the μ-cluster. Users select the decay function to use assigning the corresponding string to the `forget_method` parameter.

To illustrate how the forgetting process changes the data representation see the example illustrated in Figure B.1. This figures represent the accumulated data samples at different time instants. Let's analyze this data as the payment profiles of the a bank credit card users. In the x axis the debt of the customers is represented and in the y axis their income. At $t = 0$ the class in red are those users with low debt to income (DtoI) ratio. The blue class are those user with medium DtoI ratio and the green ones are those credit users whit limit DtoI ratio.



(a) $t = 30$ (b) $t = 390$ (c) $t = 660$ (d) $t = 900$

Figure B.1: Snapshots of data samples for 3 cluster distribution

The same data can be analyzed in different ways aiming different objectives. For example, to implement new marketing strategies the marketing branch of a bank may be interested in the whole data history. To the contrary, the credit risk department will be specially focused in the current users behavior which might quickly becomes hazardous for the bank. The

clusters found by ***DyClee*** representing the users behavior, without and with the forgetting process are shown in figure B.2 and B.3 respectively.

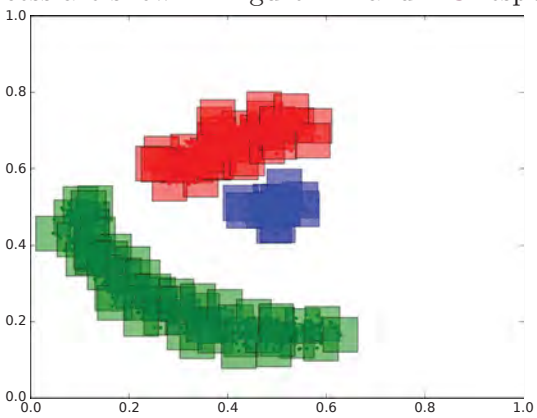


Figure B.2: ***DyClee*** clustering results with no forgetting process

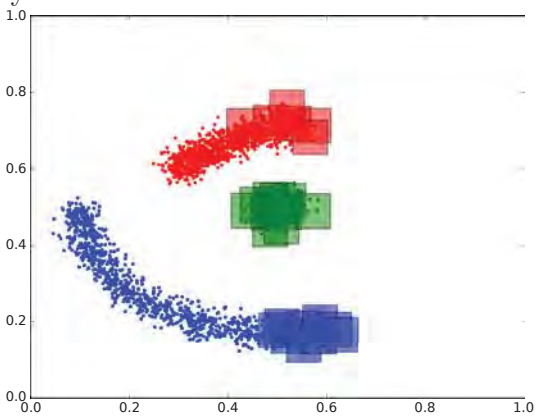


Figure B.3: ***DyClee*** clustering results with the linear function used for forgetting

B.0.4 Forgetting parameters

In section B.0.3 the mathematical functions implemented in ***DyClee*** were presented, each of these functions depend on the current time and the last assignation time, but also on specific parameters. For example, m in the case of the linear function or λ in the case of the case of the exponential function. In ***DyClee*** all these parameters are coded with only two variables: `beta` and `lamda`[1]. The relation between these parameters and the desired behavior in each of the functions is presented in Table B.1. Where $t_{\sim f}$ represents the period of time in which the μ-cluster will suffer of no ageing. The parameter $t_{w=0}$ represents the time when the forgetting function reaches zero, and accordingly $t_{w=0.5}$ and $t_{w=0.05}$ represent the time when the function reaches the 0.5 and 0.05 value respectively.

Function	`beta`	`lamda`
linear	$t_{w=0} \Leftrightarrow 1/m$	–
trapezoidal	$t_{\sim f}$	$t_{w=0}$
z function	$t_{\sim f}$	$t_{w=0}$
exponential	–	$\lambda \Leftrightarrow \frac{3}{t_{w\simeq 0.05}}$
half-life	β	λ
sigmoidal	$t_{\sim f}$	$t_{w=0.5}$

Table B.1: ***DyClee*** equivalence of coded variables `beta` and `lamda` to forgetting functions parameters

B.0.5 Long-term memory

Parameter name: `ltm`
Input type: Boolean value
Default value: `False`
Description: When true, the long-term memory is activated, *i. e.* a different forgetting factor

[1] lamda is used instead of lambda since the former is a reserved word in the implementation language Python

will be used to forget the once $\mathbb{D}\mu$Cs from those μ-cluster that have never achieved the dense status.

As stated in Chapter 6 $\mathbb{D}\mu$Cs are considered as a good representation of process behavior in a given moment of time and consequently they should not be forgotten at the same rate as the less representative μ-clusters. If the `ltm` option is activated, the once dense μ-clusters will follow a different forgetting dynamic.

If `ltm_method` is specify the *Old*μ-clusters are subject to a forgetting process following the precised forgetting function. The `ltm_method` parameter accept the same entries as the `forget_method` parameter. If `ltm_method` is not specified, the function specified in `forget_method` is also used for the long-term memory, but with a slower dynamic.

B.0.6 Non-informative label accepted

Parameter name: `Unclass_accepted`
Input type: Boolean value
Default value: `True`
Description: When true, ***DyClee*** clustering results will consider only $\mathbb{S}\mu$Cs and $\mathbb{D}\mu$Cs to find the clusters distribution. $\mathbb{O}\mu$Cs will be considered as containing non representative information and the samples represented by them will be rejected (classed as noise).

According with the use case, one can be interested in leave some abnormal data out (outlier rejection) or in the contrary to classify them. Indeed, as stated for Kempowsky *et al.*, the rejection of some samples is very important for industrial fault detection and diagnosis [Kem *et al.* 06]. Abnormal samples can corrupt the information about a system state and in the case were they are present in the learning stages they can induce a bias in the state representations that may cause misdetections.

However, in some cases, it is desirable to label all samples even if they seems abnormal. An example, described by [Pyo *et al.* 11], is that of credit card issuing process for renewal, where all subjects should be classed whether as *approved* or as *rejected*.

Figures B.4 and B.5 show ***DyClee*** clustering results in a concentric circles data set when the `Unclass_accepted` parameter is set to `True` and `False` respectively. Two classes are detected marked in red and blue colors. In addition, Figure B.4 shows some gray μ-clusters corresponding to abnormal samples. In Figure B.5 these samples are classed according to the closest cluster.

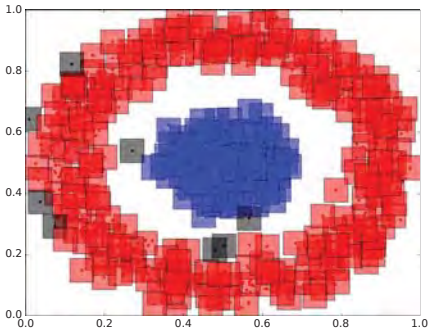


Figure B.4: ***DyClee*** clustering results with `Unclass_accepted` True

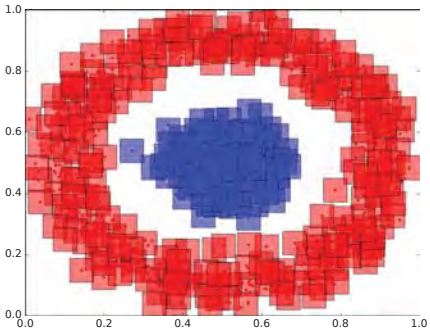


Figure B.5: ***DyClee*** clustering results with the `Unclass_accepted` False

B.0.7 Minimum amount of μ-clusters

Parameter name: `minimum_mc`
Input type: Boolean value
Default value: `False`
Description: This option allows the user to reject clusters describing a small amount of objects.

This option is highly desirable when the amount of objects described by each cluster are on the same scale, since they can facilitate the process of finding clusters that are not strictly dense connected. On the other hand, this option could not be desirable in some situations, as is the case of fault diagnosis. As said before, fault diagnosis is desirable as soon as possible. In faulty conditions the amount of information collected of the faulty state is always less than the amount of information available of the normal behavior. In those conditions, having a restriction in the minimum number of elements to form a cluster could lead to late fault detection and even the non-detection of the fault.

To illustrate the clustering results of ***DyClee*** when this option is activated, a three cluster example is used. Figure B.6 shows the samples clustering when `minimum_mc` is set to `True` and Figure B.7 when it is set to `False`.

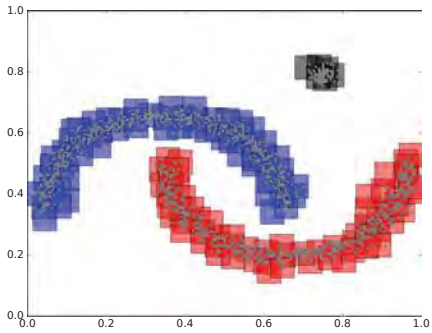


Figure B.6: ***DyClee*** clustering results with `minimum_mc` True

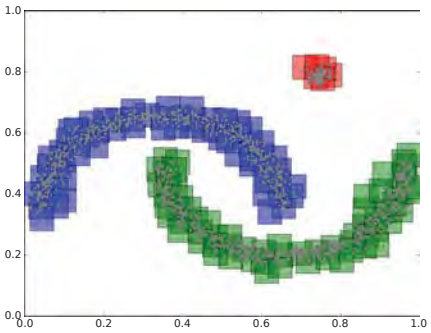


Figure B.7: ***DyClee*** clustering results with the `minimum_mc` False

B.0.8 Multi-density Clustering

Parameter name: `multi-density`
Input type: Boolean value
Default value: `False`
Description: When activated density thresholds are found locally using the clusters densities. When deactivated, a global density threshold is used.

DyClee allows detection of clusters having densities in different density levels as introduced in chapter 6. This multi-density detection is posible when the local-based analysis is performed. This analysis is optional in ***DyClee***, and can be set on or off according with user needs. Interestingly, using the multi-density scheme helps to better shape clusters that share frontiers, that is, clusters that are side by side, in highly overlapping situations. To do this, ***DyClee*** first finds all possible clusters, allowing multiple densities. Second, it finds the borders shared by two or more clusters and then analyzes every μ-cluster in those borders. These μ-cluster are assigned to the connected cluster that has the most similar average density. In that way clusters frontiers can be precisely drawn which is a key feature in highly overlapping distributions.

Figures B.8 and B.9 show a multidensity distribution clustered by ***DyClee*** with the local-density analysis in the former and global-density analysis in the later. In Figure B.9 only the denser areas of the clusters are detected loosing its relation with the other near points.

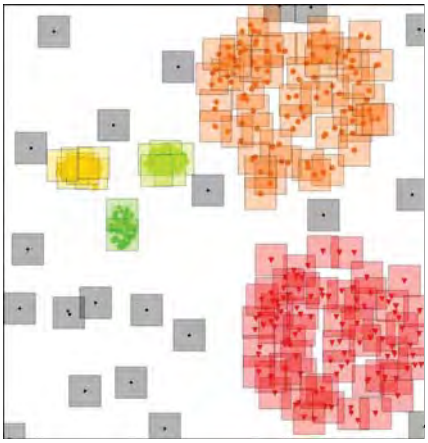


Figure B.8: ***DyClee*** clustering results with `multi-density` True

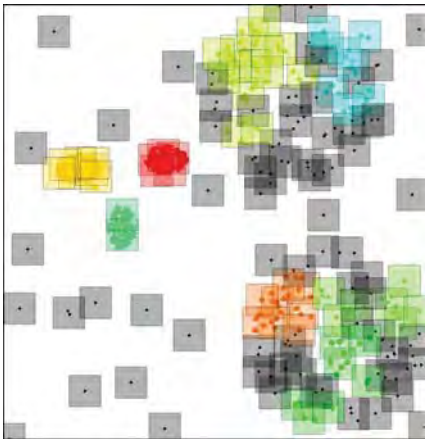


Figure B.9: ***DyClee*** clustering results with `multi-density` False

B.0.9 Time between density-based analyses clustering

Parameter name: `t_off`
Input type: Real value
Default value: When the whole data set is available at once is a quarter of the total amount of elements. When data is coming in stream is set to 1000 samples.
Description: Time between two runs of the density-based clustering algorithm. This time establish the maximum amount of time before labeling.

The density-based stage runs in parallel to the distance-based stage of ***DyClee*** but its complexity is higher. Since usually new clusters are a rare event there is no need for this stage to run all the time. In supervision, for example, the time between two runs of this stage should correspond to the maximum amount of time before detection of a new behavior. The density-based stage performs also the purge of those $\mathbb{O}\mu$Cs with extremely low densities, therefore, this time should be choose also thinking in the release of memory and CPU resources.

B.0.10 Dynamic window parameters

Three parameters can be configured to force ***DyClee*** dynamic window splitting. These parameters are:

- Minimum window size (`Min_ws`): Integer value. The minimum window size is related to the length of non linear episodes, that is, the length of the shortest non linear behavior described by an episode expressed in number of samples. When known this value improves the quality of the polynomial approximation mostly in highly noisy environments. The default value is 100 (samples).

- Maximum window size (`Max_ws`): Integer value. The length of the longest known linear behavior measured as the amount of samples belonging to the longest linear episode. The default value is 1000 (samples).

- Normal window size (`ws`): Integer value. The initial window length to be considered by the preprocessing stage. Default value is 200 (samples).

Bibliography

[Agg and Red 13] C. C. Aggarwal and C. K. Reddy. Data clustering: algorithms and applications. CRC Press, 2013.

[Agg 14] C. C. Aggarwal. Data classification: algorithms and applications. CRC Press, 2014.

[Agg *et al.* 99] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. *Fast Algorithms for Projected Clustering.* ACM SIGMOD Record, vol. 28, no. 2, pages 61–72, June 1999.

[Agg *et al.* 01] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space.* In Database Theory - ICDT 2001, J. Van den Bussche and V. Vianu, editors, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer Berlin Heidelberg, 2001.

[Agg *et al.* 03] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. *A framework for clustering evolving data streams.* In Proceedings of the 29th international conference on Very large data bases-Volume 29, pages 81–92. VLDB Endowment, 2003.

[Agu and Man 82] J. Aguilar-Martín and R. Lopez de Mantaras. *The process of classification and learning the meaning of linguistic descriptors of concepts.* In Approximate reasoning in decision analysis, M. M. Gupta and E. Sánchez, editors, pages 165–175. North-Holland Pub. Co., 1982.

[Agu *et al.* 12] J. Aguilar, A. Subias, L. Travé-Massuyès, and K. Zouaoui. *Situation assessment in autonomous systems.* In 2012 Global Information Infrastructure and Networking Symposium (GIIS), pages 1–6, Dec 2012.

[Alc and Man 99] R. J. Alcock and Y. Manolopoulos. *Time-series similarity queries employing a feature-based approach.* In Proceedings of the 7 th Hellenic Conference on Informatics, 1999.

[Alu and Dil 94] R. Alur and D. L. Dill. *A theory of timed automata.* Theoretical Computer Science, vol. 126, no. 2, pages 183 – 235, 1994.

[Amo and Mir 12] R. C. de Amorim and B. Mirkin. *Minkowski metric, feature weighting and anomalous cluster initializing in K-Means clustering.* Pattern Recognition, vol. 45, no. 3, pages 1061 – 1075, 2012.

[Ana and Geo 01] G. C. Anagnostopoulos and M. Georgiopoulos. *Ellipsoid ART and ARTMAP for incremental unsupervised and supervised learning.* In Aerospace/Defense Sensing, Simulation, and Controls., volume 4390, pages 293–304. International Society for Optics and Photonics., 2001.

[Ang and Zho 08] P. Angelov and X. Zhou. *Evolving Fuzzy-Rule-Based Classifiers From Data Streams.* Fuzzy Systems, IEEE Transactions on, vol. 16, no. 6, pages 1462–1475, 2008.

[Ang 01] L. Angstenberger. *Dynamic Fuzzy Pattern Recognition with Applications to Finance and Engineering.* PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 2001.

[Ang 11] P. Angelov. *Fuzzily Connected Multimodel Systems Evolving Autonomously From Data Streams.* Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 41, no. 4, pages 898–910, 2011.

[Apa 16a] Apache Hadoop. *Hadoop*, 2016. http://hadoop.apache.org Last consulted 2016-08-30.

[Apa 16b] Apache Kafka. *A high-throughput, distributed messaging system*, 2016. http://kafka.apache.org Last consulted 2016-08-30.

[Bar *et al.* 13] A. Barrios, S. Panche, M. Duque, V. H. Grisales, F. Prieto, J. L. Villa, P. Chevrel, and M. Canu. *A multi-user remote academic laboratory system.* Computers & Education, vol. 62, pages 111 – 122, 2013.

[Bar *et al.* 15] N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *Trend-Based Dynamic Classification for on-line Diagnosis of Time-Varying Dynamic Systems.* In SAFEPROCESS 2015, Proceedings of the 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, pages 1224–1231. IFAC, 2015.

[Bar *et al.* 16a] N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *DyClee: Dynamic clustering for tracking evolving environments.* Pattern Recognition, 2016. Submitted.

[Bar *et al.* 16b] N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *Dynamic Clustering as a Tool for Monitoring Evolving Systems.* In DX-2016, Proceedings of the 27th International Workshop on Principles of Diagnosis. DX, 2016.

[Bar *et al.* 16c] N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *Dynamic clustering for process supervision.* In XVII CLCA, Latin American Conference of Automatic Control. IFAC, 2016.

[Bar *et al.* 16d] N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales. *A novel algorithm for dynamic clustering: properties and performance.* In ICMLA 2016, 15th IEEE International Conference on Machine Learning and Applications. IEEE, 2016.

[Bec *et al.* 90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles.* In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, SIGMOD '90, pages 322–331, New York, NY, USA, 1990. ACM.

[Ben 75] J. L. Bentley. *Multidimensional binary search trees used for associative searching.* Communications of the ACM, vol. 18, no. 9, pages 509–517, 1975.

[Ben *et al.* 99] K. P. Bennett, U. Fayyad, and D. Geiger. *Density-based indexing for approximate nearest-neighbor queries.* In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 233–243. ACM, 1999.

[Ber and Hül 06] J. Beringer and E. Hüllermeier. *Online clustering of parallel data streams.* Data & Knowledge Engineering, vol. 58, no. 2, pages 180 – 204, 2006.

[Bey *et al.* 99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. *When Is "Nearest Neighbor" Meaningful?* In Database Theory — ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings, C. Beeri and P. Buneman, editors, pages 217–235, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Bez *et al.* 84] J. C. Bezdek, R. Ehrlich, and W. Full. *FCM: The fuzzy c-means clustering algorithm.* Computers & Geosciences, vol. 10, no. 2, pages 191–203, 1984.

[Bla 03] M. Blanke. Diagnosis and fault-tolerant control. Springer Verlag, 2003.

[Bob and Bez 91] L. Bobrowski and J. C. Bezdek. *c-means clustering with the l_1 and l_∞ norms.* IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 3, pages 545–554, May 1991.

[Bor and Bha 07] B. Borah and D. K. Bhattacharyya. *A Clustering Technique using Density Difference.* In 2007 International Conference on Signal Processing, Communications and Networking, pages 585–588, 2007.

[Bot *et al.* 13] J. F. Botía, C. Isaza, T. Kempowsky, M. V. L. Lann, and J. Aguilar-Martín. *Automaton based on fuzzy clustering methods for monitoring industrial processes.* Engineering Applications of Artificial Intelligence, vol. 26, no. 4, pages 1211 – 1220, 2013.

[Bou and Van 11] A. Bouchachia and C. Vanaret. *Incremental learning based on growing gaussian mixture models.* In Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on, volume 2, pages 47–52. IEEE, Elsevier, 2011.

[Bou 09] A. Bouchachia. *Incremental Learning.* In Encyclopedia of Data Warehousing and Mining, Second Edition, pages 1006–1012. IGI Global, 2009.

[Bou 11] A. Bouchachia. *Fuzzy classification in dynamic environments.* Soft Computing, vol. 15, no. 5, pages 1009–1022, 2011.

[Cam and Alc 15] Camara de Comercio de Bogotá and Alcaldía mayor de Bogotá. *Bogotá Escenarios 2025*, 2015.

[Cam 11] Camara de Comercio de Bogotá. *Boletín Cifras. Dirección de Estudios e Investigaciones*, October 2011. Vicepresidencia de Gestión pública y social.

[Cam 15] Camara de Comercio de Bogotá. *Observatorio de la región Bogotá - Cundinamarca. Dinámica de la economía y el comercio exterior de la región.* Observatorio de la región Bogotá - Cundinamarca, no. 24, March 2015.

[Cao *et al.* 06] F. Cao, M. Ester, W. Qian, and A. Zhou. *Density-based Clustering over an Evolving Data Stream with Noise.* In SDM, pages 326–337, 2006.

[Car and Gro 10] G. A. Carpenter and S. Grossberg. Adaptive resonance theory, In Encyclopedia of Machine Learning, C. Sammut and G. Webb, editors, pages 22 – 35. Springer, 2010.

[Car *et al.* 92] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. *Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps.* IEEE Transactions on Neural Networks, vol. 3, no. 5, pages 698–713, Sep 1992.

[Cas and Laf 10] C. G. Cassandras and S. Lafortune. Introduction to discrete event systems. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[Cha and Yeu 08] H. Chang and D.-Y. Yeung. *Robust path-based spectral clustering.* Pattern Recognition, vol. 41, no. 1, pages 191–203, 2008.

[Che and Ste 90] J.-Y. Cheung and G. Stephanopoulos. *Representation of process trends–Part I. A formal representation framework.* Computers & Chemical Engineering, vol. 14, no. 4, pages 495 – 510, 1990.

[Che and Tu 07] Y. Chen and L. Tu. *Density-based clustering for real-time stream data.* In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and data mining, pages 133–142, 2007.

[Che *et al.* 12] X. Chen, J. Huang, Y. Wang, and C. Tao. *Incremental feedback learning methods for voice recognition based On DTW.* In Modelling, Identification Control (ICMIC), 2012 Proceedings of International Conference on, pages 1011–1016, June 2012.

[Chi *et al.* 01] L. Chiang, E. Russell, and R. Braatz. Fault detection and diagnosis in industrial systems. Springer Verlag, 2001.

[Cor *et al.* 07] M.-O. Cordier, A. Grastien,*et al. Exploiting Independence in a Decentralised and Incremental Approach of Diagnosis.* In IJCAI, pages 292–297, 2007.

[Das and Nin 00] D. Dasgupta and F. Nino. *A comparison of negative and positive selection algorithms in novel pattern detection.* In Systems, Man, and Cybernetics, 2000 IEEE International Conference on, volume 1, pages 125–130 vol.1, 2000.

[Das *et al.* 04] S. Dash, M. R. Maurya, V. Venkatasubramanian, and R. Rengaswamy. *A novel interval-halving framework for automated identification of process trends.* AIChE journal, vol. 50, no. 1, pages 149–162, 2004.

[Dav *et al.* 07] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. *Information-theoretic Metric Learning.* In Proceedings of the 24th International Conference on Machine Learning, ICML '07, pages 209–216, New York, NY, USA, 2007. ACM.

[Din 08] S. Ding. Model-based fault diagnosis techniques: design schemes, algorithms, and tools. Springer Verlag, 2008.

[Din 14] S. Ding. Data-driven design of fault diagnosis and fault-tolerant control systems. Springer Verlag, 2014.

[Dje *et al.* 09] M. Djeziri, B. O. Bouamama, and R. Merzouki. *Modelling and robust FDI of steam generator using uncertain bond graph model.* Journal of Process Control, vol. 19, no. 1, pages 149 – 162, 2009.

[Doh *et al.* 04] K. Doherty, R. Adams, and N. Davey. *Non-Euclidean norms and data normalisation.* In Proceedings of the European Symposium on Artificial Neural Networks (ESANN), pages 181–186, 2004.

[Eln *et al.* 07] S. Elnekave, M. Last, and O. Maimon. *Incremental Clustering of Mobile Objects.* In Data Engineering Workshop, 2007 IEEE 23rd International Conference on, pages 585–592, April 2007.

[Est *et al.* 96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. *A density-based algorithm for discovering clusters in large spatial databases with noise.* In KDD, pages 226–231, 1996.

[Est *et al.* 01] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski. *Algorithmic Transformations in the Implementation of K- Means Clustering on Reconfigurable Hardware.* In Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, FPGA '01, pages 103–110, New York, NY, USA, 2001. ACM.

[Fah *et al.* 10] A. Fahim, A.-E. Salem, F. Torkey, M. Ramadan, G. Saake,*et al. Scalable Varied Density Clustering Algorithm for Large Datasets.* Journal of Software Engineering and Applications, vol. 3, no. 06, page 593, 2010.

[Fan *et al.* 08] Y. Fang, M. Friedman, G. Nair, M. Rys, and A.-E. Schmid. *Spatial Indexing in Microsoft SQL Server 2008.* In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, pages 1207–1216, New York, NY, USA, 2008. ACM.

[Fil and Tse 06] D. Filev and F. Tseng. *Novelty Detection Based Machine Health Prognostics.* In Evolving Fuzzy Systems, 2006 International Symposium on, pages 193–199, 2006.

[Fis *et al.* 01] B. Fischer, T. Zöller, and J. Buhmann. *Path Based Pairwise Data Clustering with Application to Texture Segmentation.* In Energy Minimization Methods in Computer Vision and Pattern Recognition, M. Figueiredo, J. Zerubia, and A. Jain, editors, volume 2134 of *Lecture Notes in Computer Science*, pages 235–250. Springer Berlin Heidelberg, 2001.

[Fra *et al.* 07] D. Francois, V. Wertz, and M. Verleysen. *The Concentration of Fractional Distances.* IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 7, pages 873–886, July 2007.

[GA 12] F. I. Gamero Argüello. *Pattern recognition based on qualitative representation of signals. Application to situation assessment of dynamic systems.* PhD thesis, Universitat de Girona, June 2012.

[GA *et al.* 09] F. I. Gamero Argüello, J. Colomer, J. Meléndez, and X. Berjaga. *Qualtras: a Tool for Qualitative Trend Representations.* DX-09 June 14-17 Stockholm, page 187, 2009.

[Gam *et al.* 14a] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. *A survey on concept drift adaptation.* ACM Computing Surveys (CSUR), vol. 46, no. 4, page 44, 2014.

[Gam *et al.* 14b] F. I. Gamero, J. Meléndez, and J. Colomer. *Process diagnosis based on qualitative trend similarities using a sequence matching algorithm.* Journal of Process Control, vol. 24, no. 9, pages 1412 – 1424, 2014.

[Gar 10] D. Garcia. *Robust smoothing of gridded data in one and higher dimensions with missing values.* Computational statistics & data analysis, vol. 54, no. 4, pages 1167–1178, 2010.

[Gau *et al.* 15] Q. Gaudel, E. Chanthery, and P. Ribot. *Hybrid Particle Petri Nets for Systems Health Monitoring under Uncertainty.* International Journal of Prognostics and Health Management, vol. 6, June 2015.

[Ger 88] J. Gertler. *Survey of model-based failure detection and isolation in complex plants.* Control Systems Magazine, IEEE, vol. 8, no. 6, pages 3 –11, dec 1988.

[Gio *et al.* 07] A. Gionis, H. Mannila, and P. Tsaparas. *Clustering Aggregation.* ACM Trans. Knowl. Discov. Data, vol. 1, no. 1, March 2007.

[Gou *et al.* 13] R. Gouriveau, E. Ramasso, and N. Zerhouni. *Strategies to face imbalanced and unlabelled data in PHM applications.* Chemical Engineering Transactions, vol. 33, pages 115–120, 2013.

[Gus and Kes 78] D. E. Gustafson and W. C. Kessel. *Fuzzy clustering with a fuzzy covariance matrix.* In Decision and Control including the 17th Symposium on Adaptive Processes, 1978 IEEE Conference on, pages 761–766, Jan 1978.

[Haa and P¸10] B. Haasdonk and E. Pękalska. *Classification with Kernel Mahalanobis Distance Classifiers.* In Advances in Data Analysis, Data Handling and

Business Intelligence, A. Fink, B. Lausen, W. Seidel, and A. Ultsch, editors, Studies in Classification, Data Analysis, and Knowledge Organization, pages 351–361. Springer Berlin Heidelberg, 2010.

[Hat *et al.* 00] R. J. Hathaway, J. C. Bezdek, and Y. Hu. *Generalized fuzzy c-means clustering strategies using Lp norm distances*. IEEE Transactions on Fuzzy Systems, vol. 8, no. 5, pages 576–582, Oct 2000.

[Hed *et al.* 10] L. Hedjazi, T. Kempowsky-Hamon, L. Despènes, M. Le Lann, S. Elgue, and J. Aguilar-Martín. *Sensor placement and fault detection using an efficient fuzzy feature selection approach*. In Decision and Control (CDC), 2010 49th, IEEE Conference on, pages 6827 –6832, Dec. 2010.

[He *et al.* 09] X. He, Y. Yang, and D. Li. *Class-incremental fisher discriminant analysis with principal component analysis for process monitoring*. In Asian Control Conference, 2009. ASCC 2009. 7th, pages 830–834, Aug 2009.

[Hin *et al.* 00] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. *What Is the Nearest Neighbor in High Dimensional Spaces?* In Proceedings of the 26th International Conference on Very Large Data Bases, VLDB '00, pages 506–515, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Hou *et al.* 10] M. E. Houle, H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Can shared-neighbor distances defeat the curse of dimensionality?, In Scientific and Statistical Database Management: 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30–July 2, 2010. Proceedings, M. Gertz and B. Ludäscher, editors, pages 482–500. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Hub and Ara 85] L. Hubert and P. Arabie. *Comparing partitions*. Journal of Classification, vol. 2, no. 1, pages 193–218, 1985.

[Isa *et al.* 09] C. Isaza, A. Orantes Molina, T. Kempowsky, M. Le Lann,*et al. Contribution of fuzzy classification for the diagnosis of complex systems*. In Fault Detection, Supervision and Safety of Technical Processes, pages 1132–1137, 2009.

[Ise 05] R. Isermann. *Model-based fault-detection and diagnosis-status and applications*. Annual Reviews in control, vol. 29, no. 1, pages 71–85, 2005.

[Jan and Ven 91] M. Janusz and V. Venkatasubramanian. *Automatic generation of qualitative descriptions of process trends for fault detection and diagnosis*. Engineering Applications of Artificial Intelligence, vol. 4, no. 5, pages 329–339, 1991.

[Jay and Muh 14] S. A. Jayousi and M. S. Muhammad. *A Survey of Contact Testing Techniques for the Diagnosis of Printed Circuit Boards*. International Journal of Scientific & Engineering Research, vol. 5, no. 4, 2014.

[Jim *et al.* 11] R. Jimenez, F. Prieto, and V. H. Grisales. *Detection of the Tiredness Level of Drivers Using Machine Vision Techniques*. In Electronics, Robotics and

Automotive Mechanics Conference (CERMA), 2011 IEEE, pages 97–102, Nov 2011.

[Kae *et al.* 96] L. P. Kaelbling, M. L. Littman, and A. W. Moore. *Reinforcement learning: A survey.* Journal of artificial intelligence research, pages 237–285, 1996.

[Kar *et al.* 99a] G. Karypis, E.-H. Han, and V. Kumar. *Chameleon: A Hierarchical clustering Algorithm using Dynamic Modeling.* Technical Report 99-007, University of Minnesota - Department of Computer Science and Engineering, 4-192 EECS Bldg., 200 Union St. SE Minneapolis, MN 55455, USA, 1999.

[Kar *et al.* 99b] G. Karypis, E.-H. Han, and V. Kumar. *Chameleon: Hierarchical clustering using dynamic modeling.* Computer, vol. 32, no. 8, pages 68–75, 1999.

[Kem 04] T. Kempowsky. *Surveillance de procédés à base de méthodes de classification: conception d'un outil d'aide pour la détection et le diagnostic des défaillances.* PhD thesis, INSA de Toulouse, 2004.

[Kem *et al.* 06] T. Kempowsky, A. Subias, and J. Aguilar-Martín. *Process situation assessment: From a fuzzy partition to a finite state machine.* Engineering Applications of Artificial Intelligence, vol. 19, no. 5, pages 461–477, 2006.

[Keo and Paz 00] E. Keogh and M. Pazzani. *Scaling up dynamic time warping for data mining applications.* In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 285–289. ACM, 2000.

[Kil *et al.* 11] B. Kilundu, P. Dehombreux, and X. Chiementin. *Tool wear monitoring by machine learning techniques and singular spectrum analysis.* Mechanical Systems and Signal Processing, vol. 25, no. 1, pages 400–415, 2011.

[Koy 00] I. Koychev. *Gradual forgetting for adaptation to concept drift.* In Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning, pages 101–106. ECAI, 2000.

[Kra *et al.* 11] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. *The ClusTree: indexing micro-clusters for any stream mining.* Knowledge and information systems, vol. 29, no. 2, pages 249–272, 2011.

[Kri *et al.* 09] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. *Outlier detection in axis-parallel subspaces of high dimensional data.* In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 831–838. Springer, 2009.

[Kri *et al.* 11] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. *Density-based clustering.* Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 1, no. 3, pages 231–240, 2011.

[Kwa *et al.* 15] J. Kwak, T. Lee, and C. O. Kim. *An Incremental Clustering-Based Fault Detection Algorithm for Class-Imbalanced Process Data.* IEEE Transactions on Semiconductor Manufacturing, vol. 28, no. 3, pages 318–328, Aug 2015.

[Li and Pen 08] J. Li and L. Peng. *Human expression recognition based on feature block 2DPCA and Manhattan distance classifier.* In 7th World Congress on Intelligent Control and Automation, 2008. WCICA 2008., pages 5941 – 5945, June 2008.

[Lic 13] M. Lichman. *UCI Machine Learning Repository*, 2013.

[Li *et al.* 11] K. Li, F. Yao, and R. Liu. *An online clustering algorithm.* In Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on, volume 2, pages 1104–1108. IEEE, 2011.

[Lun and Sup 02] J. Lunze and P. Supavatanakul. *Diagnosis of discrete–event system described by timed automata.* IFAC Proceedings Volumes, vol. 35, no. 1, pages 77–82, 2002.

[Mac 67] J. MacQueen. *Some methods for classification and analysis of multivariate observations.* In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability Volume 1: Statistics, pages 281–297. University of California Press, 1967.

[Mao and Jai 96] J. Mao and A. K. Jain. *A self-organizing network for hyperellipsoidal clustering (HEC).* IEEE Transactions on Neural Networks, vol. 7, no. 1, pages 16–29, Jan 1996.

[Mar and Sin 03] M. Markou and S. Singh. *Novelty detection: a review–part 1: statistical approaches and part 2: neural network based approaches.* Signal processing, vol. 83, no. 12, pages 2481–2497, 2003.

[Mau *et al.* 03] M. R. Maurya, R. Rengaswamy, and V. Venkatasubramanian. *Qualitative trend analysis of the principal components: application to fault diagnosis.* In 8th International Symposium on Process Systems Engineering, B. Chen and A. W. Westerberg, editors, volume 15 of *Computer Aided Chemical Engineering*, pages 968 – 973. Elsevier, 2003.

[Mau *et al.* 05] M. R. Maurya, R. Rengaswamy, and V. Venkatasubramanian. *Fault Diagnosis by Qualitative Trend Analysis of the Principal Components.* Chemical Engineering Research and Design, vol. 83, no. 9, pages 1122 – 1132, 2005.

[Mau *et al.* 07] M. R. Maurya, R. Rengaswamy, and V. Venkatasubramanian. *Fault diagnosis using dynamic trend analysis: A review and recent developments.* Engineering Applications of Artificial Intelligence, vol. 20, no. 2, pages 133 – 146, 2007. Special Issue on Applications of Artificial Intelligence in Process Systems Engineering.

[Mau *et al.* 10] M. R. Maurya, P. K. Paritosh, R. Rengaswamy, and V. Venkatasubramanian. *A framework for on-line trend extraction and fault diagnosis.* Engineering Applications of Artificial Intelligence, vol. 23, no. 6, pages 950 – 960, 2010.

[Maz *et al.* 11] J. Mazel, P. Casas, Y. Labit, and P. Owezarski. *Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection.* In Proceedings of the 7th International Conference on Network and Services Management, pages 73–80. International Federation for Information Processing, 2011.

[ME *et al.* 14] A. Morales-Esteban, F. Martínez-Álvarez, S. Scitovski, and R. Scitovski. *A fast partitioning algorithm using adaptive Mahalanobis clustering with application to seismic zoning.* Computers & Geosciences, vol. 73, pages 132 – 141, 2014.

[Meg *et al.* 05] V. Megalooikonomou, Q. Wang, G. Li, and C. Faloutsos. *A multiresolution symbolic representation of time series.* In 21st International Conference on Data Engineering (ICDE'05), pages 668–679, April 2005.

[Mil *et al.* 94] R. Milne, C. Nicol, M. Ghallab, L. Travé-Massuyès, K. Bousson, C. Dousson, J. Quevedo, J. Aguilar, and A. Guasch. *TIGER: real-time situation assessment of dynamic systems.* Intelligent Systems Engineering, vol. 3, no. 3, pages 103 –124, Autumn 1994.

[Mit *et al.* 03] P. Mitra, S. K. Pal, and M. A. Siddiqi. *Non-convex clustering using expectation maximization algorithm with rough set initialization.* Pattern Recognition Letters, vol. 24, no. 6, pages 863 – 873, 2003.

[Mon *et al.* 15] T. Monrousseau, L. Travé-Massuyès, and M.-V. L. Lann. *Processing measure uncertainty into fuzzy classifier.* In 26th International Workshop on Principles of Diagnosis, Aug. 2015.

[Omo 89] S. M. Omohundro. *Five Balltree Construction Algorithms.* Technical report, International Computer Science Institute Berkeley, 1989.

[Ora *et al.* 07] A. Orantes, T. Kempowsky, M. Le Lann, L. Prat, S. Elgue, C. Gourdon, and M. Cabassud. *Selection of sensors by a new methodology coupling a classification technique and entropy criteria.* Chemical Engineering Research and Design, vol. 8, no. 6, pages 825–836, 2007.

[Oul 14] B. Ould Bouamama. *Contrôle en ligne d'une installation de générateur de vapeur par Bond Graph.* Techniques de l'ingénieur Méthodes de production, vol. base documentaire : TIB521DUO., no. ref. article : ag3551, July 2014. article de base documentaire.

[Pat and Dan 94] M. Paterson and V. Dančík. *Longest common subsequences.* In International Symposium on Mathematical Foundations of Computer Science, pages 127–142. Springer, 1994.

[Ped *et al.* 11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, vol. 12, pages 2825–2830, 2011.

[Pen and Cor 05] Y. Pencolé and M.-O. Cordier. *A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks.* Artificial Intelligence, vol. 164, no. 1, pages 121–170, 2005.

[Pyo *et al.* 11] C. U. Pyon, J. Y. Woo, and S. C. Park. *Service improvement by business process management using customer complaints in financial service industry.* Expert Systems with Applications, vol. 38, no. 4, pages 3267 – 3279, 2011.

[Ram and Gou 14] E. Ramasso and R. Gouriveau. *Remaining useful life estimation by classification of predictions based on a neuro-fuzzy system and theory of belief functions.* IEEE Transactions on Reliability, vol. 63, no. 2, pages 555–566, 2014.

[Ren *et al.* 01] R. Rengaswamy, T. Hägglund, and V. Venkatasubramanian. *A qualitative shape analysis formalism for monitoring control loop performance.* Engineering Applications of Artificial Intelligence, vol. 14, no. 1, pages 23 – 33, 2001.

[Rib and Bar 11] L. Ribeiro and J. Barata. *Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging {IT} based production paradigms.* Computers in Industry, vol. 62, no. 7, pages 639 – 659, 2011.

[Rou 87] P. Rousseeuw. *Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis.* J. Comput. Appl. Math., vol. 20, no. 1, pages 53–65, November 1987.

[Rub *et al.* 01] Y. Rubner, J. Puzicha, C. Tomasi, and J. M. Buhmann. *Empirical Evaluation of Dissimilarity Measures for Color and Texture.* Computer Vision and Image Understanding, vol. 84, no. 1, pages 25 – 43, 2001.

[Sak *et al.* 05] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. *FTW: Fast Similarity Search Under the Time Warping Distance.* In Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '05, pages 326–337, 2005.

[Sam *et al.* 96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. *Failure diagnosis using discrete-event models.* IEEE transactions on control systems technology, vol. 4, no. 2, pages 105–124, 1996.

[Sil and Jon 89] B. W. Silverman and M. C. Jones. *E. Fix and J.L. Hodges (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951).* International Statistical Review, vol. 57, no. 3, pages 233–238, 1989.

[Sou and Car 04] R. M. C. R. de Souza and F. A. T. de Carvalho. *Dynamic clustering of interval data based on adaptive Chebyshev distances.* Electronics Letters, vol. 40, no. 11, pages 658–660, May 2004.

[Ste *et al.* 00] M. Steinbach, G. Karypis, V. Kumar,*et al. A comparison of document clustering techniques*. In KDD workshop on text mining, volume 400, pages 525–526. Boston, 2000.

[Sto 48] M. H. Stone. *The generalized Weierstrass approximation theorem*. Mathematics Magazine, vol. 21, no. 5, pages 237–254, 1948.

[Sub *et al.* 14] A. Subias, L. Travé-Massuyès, and E. L. Corronc. *Learning chronicles signing multiple scenario instances*. {IFAC} Proceedings Volumes, vol. 47, no. 3, pages 10397 – 10402, 2014. 19th {IFAC} World Congress.

[Sup *et al.* 06] P. Supavatanakul, J. Lunze, V. Puig, and J. Quevedo. *Diagnosis of timed automata: Theory and application to the DAMADICS actuator benchmark problem*. Control Engineering Practice, vol. 14, no. 6, pages 609–619, 2006.

[Tha *et al.* 15] V. V. Thang, D. V. Pantiukhin, and A. I. Galushkin. *A Hybrid Clustering Algorithm: The FastDBSCAN*. In 2015 International Conference on Engineering and Telecommunication (EnT), pages 69–74, Nov 2015.

[Tho 15] N. Thornhill. *The CSTH web site*, 2015. Accessed: 2015-02-12. http://personal-pages.ps.ic.ac.uk/~nina/CSTHSimulation/index.htm.

[Tho *et al.* 08] N. F. Thornhill, S. C. Patwardhan, and S. L. Shah. *A continuous stirred tank heater simulation model with applications*. Journal of Process Control, vol. 18, no. 3, pages 347–360, 2008.

[TM 14] L. Travé-Massuyès. *Bridging control and artificial intelligence theories for diagnosis: A survey*. Engineering Applications of Artificial Intelligence, vol. 27, pages 1 – 16, 2014.

[Tos 11] R. Toscano. Commande et diagnostic des systèmes dynamiques. Technosoup. Ellipses, 2 edition, 2011.

[Tra and Dag 03] L. Travé-Massuyès and P. Dague. Modèles et raisonnements qualitatifs. Hermès science publications, 2003.

[Tra and Mil 97] L. Travé-Massuyès and R. Milne. *Gas-turbine condition monitoring using qualitative model-based diagnosis*. IEEE Expert, vol. 12, no. 3, pages 22 –31, May/Jun 1997.

[Tra *et al.* 04] P. Traverse, I. Lacaze, and J. Souyris. *Airbus Fly-By-Wire: A Total Approach To Dependability*. In Building the Information Society: IFIP 18th World Computer Congress Topical Sessions 22–27 August 2004 Toulouse, France, R. Jacquart, editor, pages 191–212. Springer US, Boston, MA, 2004.

[Tra *et al.* 14] V. A. Tran, O. Hirose, T. Saethang, L. A. T. Nguyen, X. T. Dang, T. K. T. Le, D. L. Ngo, M. Kubo, Y. Yamada, and K. Satou. *D-IMPACT: A Data Preprocessing Algorithm to Improve the Performance of Clustering*. Journal of Software Engineering and Applications, vol. 2014, 2014.

[Tu and Che 09] L. Tu and Y. Chen. *Stream Data Clustering Based on Grid Density and Attraction*. ACM Trans. Knowl. Discov. Data, vol. 3, no. 3, pages 12:1–12:27, July 2009.

[Tuf 11] S. Tuffery. Data mining and statistics for decision making. John Wiley & Sons, Inc., 2011.

[Tur *et al.* 14] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton. *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*, April 2014. sponsored by EMC.

[Vac *et al.* 07] G. Vachtsevanos, F. Lewis, M. Roemer, A. Hess, and B. Wu. Intelligent fault diagnosis and prognosis for engineering systems. John Wiley & Sons, Inc., March 2007.

[Vee *et al.* 02] C. J. Veenman, M. J. Reinders, and E. Backer. *A maximum variance cluster algorithm*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 24, no. 9, pages 1273–1280, 2002.

[Ven 94] V. Venkatasubramanian. *Towards Integrated Process Supervision: Current Status and Future Directions*. In Computer Software Structures Integrating Ai/kbs Systems in Process Control, K.-E. ARZEN, editor, IFAC Postprint Volume, pages 1 – 13. Pergamon, Oxford, 1994.

[Ven *et al.* 03] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. Kavuri. *A review of process fault detection and diagnosis. Part I: Quantitative model-based methods*. Computers & Chemical engineering, vol. 27, no. 3, pages 293–311, 2003.

[Vik and Jen 14] V. V. Vikjord and R. Jenssen. *Information theoretic clustering using a k-nearest neighbors approach*. Pattern Recognition, vol. 47, no. 9, pages 3070 – 3081, 2014.

[Wai *et al.* 00] J. Waissman, R. Sarrate, T. Escobet, J. Aguilar, and B. Dahhou. *Wastewater treatment process supervision by means of a fuzzy automaton model*. In Intelligent Control, 2000. Proceedings of the 2000 IEEE International Symposium on, pages 163–168. IEEE, 2000.

[Wei *et al.* 05] K. Q. Weinberger, J. Blitzer, and L. K. Saul. *Distance metric learning for large margin nearest neighbor classification*. In Advances in neural information processing systems, pages 1473–1480, 2005.

[Wu *et al.* 00] Y.-L. Wu, D. Agrawal, and A. El Abbadi. *A Comparison of DFT and DWT Based Similarity Search in Time-series Databases*. In Proceedings of the Ninth International Conference on Information and Knowledge Management, CIKM '00, pages 488–495, 2000.

[Yes *et al.* 13] M. Yesilbudak, S. Sagiroglu, and I. Colak. *A new approach to very short term wind speed prediction using k-nearest neighbor classification*. Energy Conversion and Management, vol. 69, pages 77 – 86, 2013.

[Yue 05] W.-C. Yueh. *Eigenvalues of several tridiagonal matrices.* Applied Mathematics E-Notes, vol. 5, no. 66-74, pages 210–230, 2005.

[Zha and Zho 07] M.-L. Zhang and Z.-H. Zhou. *ML-KNN: A lazy learning approach to multi-label learning.* Pattern Recognition, vol. 40, no. 7, pages 2038 – 2048, 2007.

[Zha *et al.* 97] T. Zhang, R. Ramakrishnan, and M. Livny. *BIRCH: A New Data Clustering Algorithm and Its Applications.* Data Mining and Knowledge Discovery, vol. 1, no. 2, pages 141–182, 1997.

[Zim 00] H. Zimmermann. *Dynamic fuzzy data analysis and uncertainty modeling in engineering.* In Intelligent Techniques and Soft Computing in Nuclear Science and Engineering, pages 3–15. World Scientific, 2000. Proceedings of the 4th International FLINS Conference Bruges, Belgium 28–30 August.

[Zim *et al.* 11] D. Zimpfer, P. Hattis, J. Ruppert, and D. Gavert. *Space shuttle GN & C development history and evolution.* In AIAA SPACE 2011 Conference & Exposition. American Institute of Aeronautics and Astronautics, 2011. SPACE Conferences and Exposition. Long Beach, California 27–29 September.