A General Overview of to Applications of Machine Learning on Software Engineering Problems

and

An Example related to Microservice Identification Using a Multi-Objective Genetic Algorithm

Abdelhak-Djamel Seriai

A General Overview of to Applications of Machine Learning on Software Engineering Problems

IA et ML

- « l'IA désigne la capacité à concevoir et à fabriquer des ordinateurs avec des comportements qui jusqu'à récemment, semblaient être l'apanage de l'intelligence humaine» [Andrew Moore]

- L'intelligence artificielle (IA) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine » [Larousse]

- Des technologies comme l'analyse prédictive, la modélisation et la simulation, ainsi que le Machine Learning peuvent être englobées dans l'IA.

- Deux classes d'IA
  - L'IA symbolique a pour but de reproduire le raisonnement de l'Homme, modélisé par un ensemble de symboles. Cette représentation symbolique est régentée par des règles (instructions) que l'homme va coder à la machine pour guider sa prise de décision.

  - Le machine learning exploite des algorithmes sans utiliser d'instructions explicites. Les machines acquièrent des connaissances et apprennent par elles-mêmes (apprentissage automatisé) à partir d'exemples qui leur ont été fournis. Leur compréhension et leur capacité à prendre de (bonnes) décisions s'étoffent au fur et à mesure de leurs apprentissages.

IA et ML

- IA Symbolique Versus Apprentissage automatique

  - Pour l'apprentissage automatique, les machines observent et analysent des modèles dans les données. Elles utilisent des boucles de rétroaction (effet de retour sur l'action de la machine) pour contrôler et affiner leurs prédictions.

  - À l'inverse, l'intelligence artificielle symbolique repose sur des règles qui ont été créées par l'être humain. Ces règles ont ensuite été intégrées aux machines pour les guider dans leurs prévisions. Les machines n'improvisent pas et agissent en fonction des instructions explicites qu'elles ont reçues.

- Exemples de techniques d'IA: knowledge-based approach, automated reasoning, expert systems, heuristic search strategies, temporal logic, planning, pattern recognition, …

IA et ML

- Le Machine Learning est une sous-branche de l'IA, qui consiste à créer des algorithmes capables de s'améliore automatiquement avec l'expérience. On parle également dans ce cas de systèmes auto-apprenants.

- L'apprentissage automatique (ML) est la discipline qui étudie les méthodes permettant d'inférer automatiquement des modèles à partir de données.

- Ils sont particulièrement utiles pour :
  - Les domaines problématiques mal compris où les humains disposent de peu de connaissances pour développer des algorithmes efficaces ;
  - Les domaines dans lesquels il existe de grandes bases de données contenant des régularités implicites précieuses à découvrir ;
  - Des domaines où les programmes doivent s'adapter à des conditions changeantes.

ML

- **Quelques applications de l'apprentissage automatique**
  - Recherche sur le Web : classement des pages web en fonction de ce sur quoi vous êtes le plus susceptible de cliquer.
  - Biologie computationnelle : conception rationnelle de médicament par l'ordinateur basée sur des expériences passées.
  - Finances : Évaluation du risque sur les offres de crédit. Comment décider où investir de l'argent.
  - E-commerce : Prévoir l'attrition des clients. Si une transaction est frauduleuse ou non.
  - Robotique : comment gérer l'incertitude dans de nouveaux environnements. Autonome. Voiture autonome.
  - Extraction d'informations : posez des questions sur des bases de données sur le Web.
  - Réseaux sociaux : Données sur les relations et les préférences. Apprentissage automatique pour extraire de la valeur des données.
  - Etc.

ML

- **Il existe quatre types de machine learning :**
  - Apprentissage supervisé (également appelé apprentissage inductif) : Les données d'apprentissage incluent les résultats souhaités. Les algorithmes s'appuient sur des jeux de données déjà catégorisés, afin de comprendre les critères utilisés pour la classification et de les reproduire. Il permet à la machine apprendre à classifier des données selon des critères préalablement déterminés par un humain.
    - Est le plus mature, le plus étudié et le type d'apprentissage utilisé par la plupart des algorithmes d'apprentissage automatique. Apprendre avec supervision est beaucoup plus facile que d'apprendre sans supervision.
    - On nous donne des exemples d'une fonction sous la forme de données (x) et de la sortie de la fonction (f(x)). Le but de l'apprentissage inductif est d'apprendre la fonction pour de nouvelles données (x).
    - Méthodes d'apprentissage supervisé
      - Boosting
      - Machine à vecteurs de support
      - Mélanges de lois
      - Réseau de neurones artificiels
      - Méthode des k plus proches voisins
      - Arbre de décision
      - Classification naïve bayésienne
      - Inférence grammaticale
      - Espace de versions

ML

- **Il existe quatre types de machine learning :**
  - Apprentissage non supervisé : les données d'entraînement n'incluent pas les résultats souhaités (Exemple : le clustering). Les algorithmes s'entraînent à partir de données brutes, ils essaient d'extraire des patterns. Il permet donc à la machine de classer les données brutes selon les critères qu'il détermine par lui-même. Il est difficile de dire ce qu'est un bon apprentissage et ce qui ne l'est pas. Exemple d'algorithmes d'apprentissage non supervisé :
    1. K-means clustering (K-moyenne)
    2. Dimensionality Reduction (Réduction de la dimensionnalité)
    3. Principal Component Analysis (Analyse des composants principaux)
    4. Singular Value Decomposition (Décomposition en valeur singulière)
    5. Independent Component Analysis (Analyse en composantes indépendantes)
    6. Distribution models (Modèles de distribution)
    7. Hierarchical clustering (Classification hiérarchique)
  - Apprentissage semi-supervisé : les données d'entraînement incluent quelques résultats souhaités.
  - Apprentissage par renforcement : récompenses d'une séquence d'actions. L'algorithme fonctionne comme un agent autonome, qui observe son environnement et apprend au fur et à mesure des interactions avec celui-ci.Il permet à la machine d'observer son environnement et apprendre par l'entraînement et la stimulation.

ML

- **Autres éléments pour la classification des méthodes de ML**
  - Classification : lorsque la fonction en cours d'apprentissage est discrète.
  - Régression : lorsque la fonction apprise est continue.
  - Estimation de la probabilité : lorsque la sortie de la fonction est une probabilité.

- **Le Deep Learning (apprentissage profond)**
  - Est un sous-domaine du Machine Learning, dans lequel on développe des algorithmes capables de reconnaître des concepts abstraits, à l'image d'un jeune enfant à qui l'on apprend à distinguer un chien d'un cheval.

  - Est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires.

  - Ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur, du traitement automatisé du langage.

ML

- **Éléments clés de l'apprentissage automatique :**
  - Il existe des dizaines de milliers d'algorithmes d'apprentissage automatique et des centaines de nouveaux algorithmes sont développés chaque année.

  - Chaque algorithme d'apprentissage automatique a trois composants. Tous les algorithmes d'apprentissage automatique sont des combinaisons de ces trois composants :
    - Représentation : comment représenter la connaissance. Les exemples incluent les arbres de décision, les ensembles de règles, les instances, les modèles graphiques, les réseaux de neurones, les machines à vecteurs de support, les ensembles de modèles et autres.

    - Évaluation : la manière d'évaluer les programmes candidats (hypothèses). Les exemples incluent la précision, la prédiction et le rappel, l'erreur quadratique, la probabilité, la probabilité postérieure, le coût, la marge, la divergence d'entropie k-L et autres.

    - Optimisation : la manière dont les programmes candidats sont générés, appelée processus de recherche. Par exemple optimisation combinatoire, optimisation convexe, optimisation contrainte.

Méthodes d’apprentissage

- Il existe de nombreux types de méthodes d'apprentissage, chacune ayant ses propres caractéristiques et se prêtant à certains problèmes d'apprentissage.
- Les principaux types de méthodes d’apprentissage [Mitchell] :
    - concept learning (CL),
    - decision tree (DT) learning,
    - artificial neural networks (NN),
    - Bayesian learning (BL),
    - reinforcement learning (RL),
    - genetic algorithms (GA) and genetic programming (GP),
    - instance-based learning (IBL, of which case-based reasoning, or CBR, is a popular method),
    - inductive logic programming (ILP),
    - analytical learning (AL, of which explanation-based learning, or EBL is a method), and
    - combined inductive and analytical learning (IAL).

Major types of learning methods

Type	Target function representation	Target function generation[1]	Search	Inductive bias	Algorithm[2]
AL (EBL)	Horn clauses	Eager, D + B, supervised,	Deductive reasoning	B + small set of Horn clauses	Prolog-EBG
BL	Bayesian network	Eager, supervised, D (global), explicit or implicit	Probabilistic, no explicit search	Minimum description length	MAP, BOC, Gibbs, NBC
CL	Conjunction of attribute constraints	Eager, supervised, D (global)	Version Space (VS) guided	$c \in H$	Candidate elimination
DT	Decision trees	Eager, D (global), supervised	Information gain (entropy)	Preference for small trees	ID3, C4.5, Assistant
GA GP	Bit strings, program trees	Eager, no D, unsupervised	Hill climbing (simulated evolution)	Fitness-driven	Prototypical GA/GP algorithms
IBL	Not explicitly defined	Lazy, D (local), supervised,	Statistical reasoning	Similarity to nearst neighbors	K-NN, LWR, CBR
ILP	If-then rules	Eager, supervised, D (global),	Statistical, general-to-specific	Rule accuracy, FOIL-gain, shorter clauses	SCA, FOIL, PROGOL, inv. resolution
NN	Artificial neural networks	Eager, supervised, D (global)	Gradient descent guided	Smooth interpolation between data points	Back-propagation
IAL	Determined by learning approaches used	Eager, D + B, supervised	Determined by learning approaches used	Determined by learning approaches used	KBANN, EBNN, FOCL
RL	Control strategy π^*	Eager, no D, unsupervised	Through training episodes	Actions with max. Q value	Q, TD

[1] The sets D and B refer to training data and domain theory, respectively.
[2] The algorithms listed are only representatives from different types of learning.

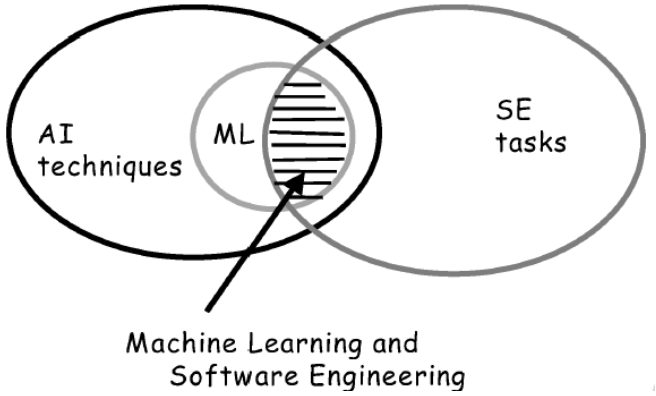
[Zhang and Tsai]

ML pour SE

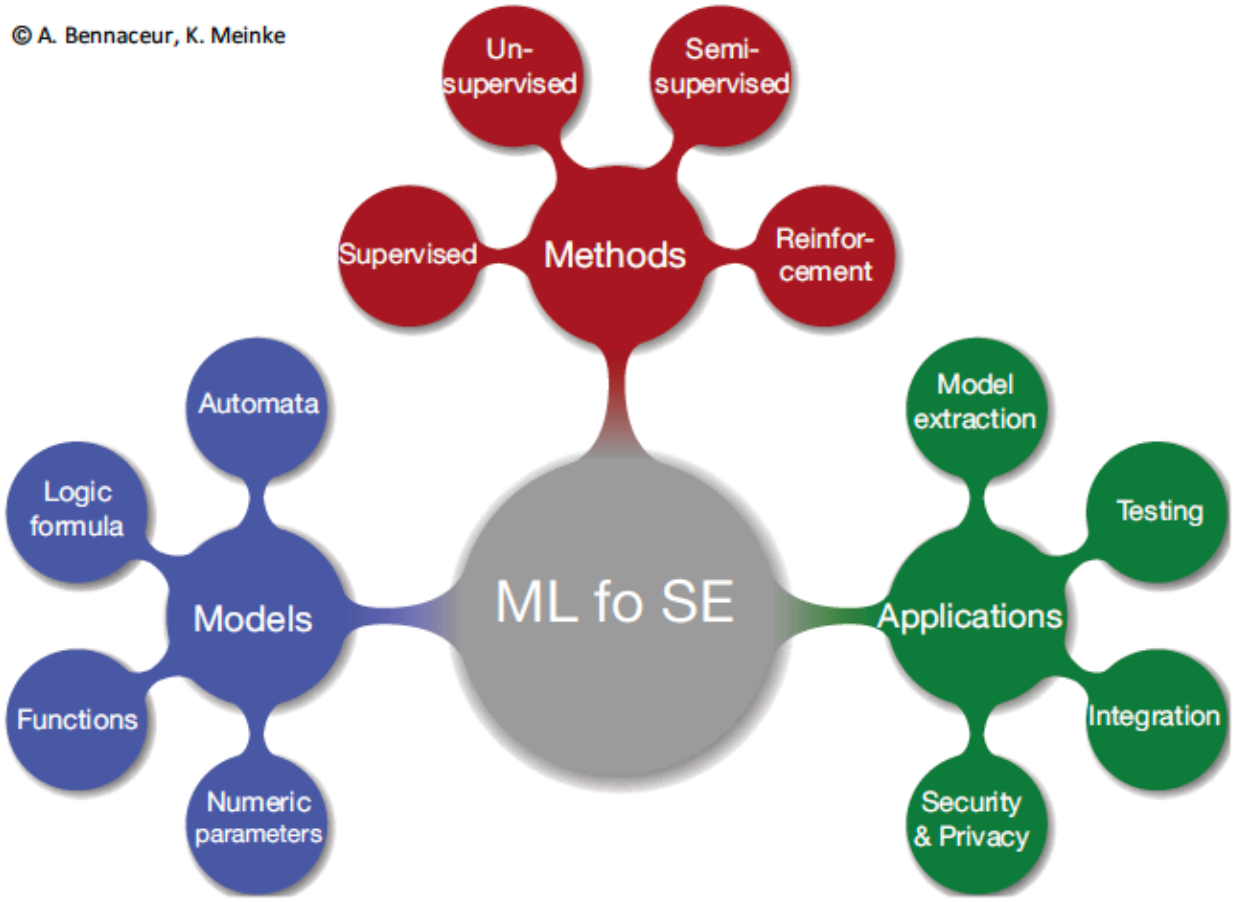
- Les algorithmes d'apprentissage automatique se sont avérés d'une grande valeur pratique dans une variété de domaines d’application.

- Le génie logiciel s'avère être un terrain fertile où de nombreuses tâches de développement et de maintenance de logiciels pourraient être formulées comme des problèmes d'apprentissage et abordées en termes d'algorithmes d’apprentissage.

- L'apprentissage automatique a été appliqué avec succès dans de nombreux domaines du génie logiciel, allant de l'extraction de comportements aux tests, en passant par la résolution de bogues.

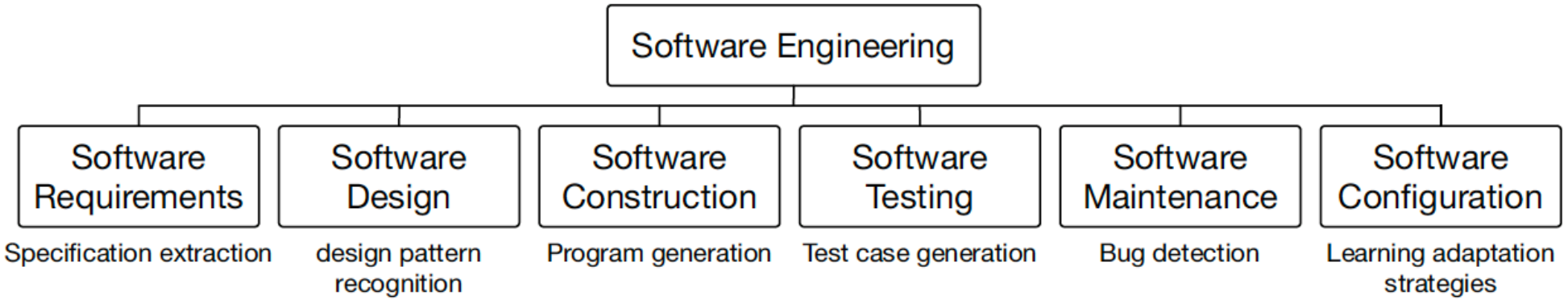


ML pour SE



Key concepts of ML for software engineering

ML pour SE



Applications of ML for software engineering

ML Pour SE : Quelques questions

- Quels types de méthodes d'apprentissage sont adaptés pour le domaine du GL ?
- Quelles sont les caractéristiques et les fondements des différents algorithmes d'apprentissage pour être appliqués aux problèmes GL?
- Comment déterminer quelle méthode d'apprentissage est appropriée pour quel type de tâche de développement ou de maintenance de logiciel ?
- Lorsque nous essayons d'utiliser une méthode d'apprentissage pour nous aider dans une tâche de GL, quelles sont les directives générales et comment pouvons-nous éviter certains pièges ?
- Quel est l'état de la pratique en matière de ML&SE ?
- etc.

ML : Procédure générale

- **Formulation/modélisation du problème (Problem formulation)**
  - La première étape consiste à formuler un problème donné de manière à ce qu'il soit conforme au cadre d'une méthode d'apprentissage particulière choisie pour la tâche.
- **Représentation du problème (Problem representation)**
  - L'étape suivante consiste à sélectionner une représentation appropriée à la fois pour les données de formation et les connaissances à apprendre.
- **Collection des données (Data collection)**
  - La troisième étape consiste à collecter les données nécessaires au processus d'apprentissage.
  - La qualité et la quantité de données nécessaires dépendent de la méthode d'apprentissage choisie. Les données peuvent nécessiter un prétraitement avant de pouvoir être utilisées dans le processus d'apprentissage.
- **Préparation du modèle théorique du domaine (Domain theory preparation)**
  - Certaines méthodes d'apprentissage (par exemple, EBL) reposent sur la disponibilité d'une théorie du domaine pour le problème donné.
- **Exécuter le processus d'apprentissage (Performing the learning process)**
  - Une fois que les données et une théorie du domaine (si nécessaire) sont prêtes, le processus d'apprentissage peut être effectué. Les données seront divisées en un ensemble d'apprentissage et un ensemble de test. Si un outil ou un environnement d'apprentissage est utilisé, les données de formation et les données de test peuvent devoir être organisées en fonction des exigences de l'outil. Les connaissances induites par l'ensemble d'apprentissage sont validées sur l'ensemble de test. En raison des différences entre l'ensemble d'apprentissage et l'ensemble de test, le processus d'apprentissage lui-même est itératif.

ML : Procédure générale

- **Analyser et évaluer les connaissances apprises (Analyzing and evaluating learned knowledge)**
  - L'analyse et l'évaluation des connaissances acquises font partie intégrante du processus d'apprentissage.
  - L'intérêt et la performance des connaissances acquises seront scrutés au cours de cette étape, souvent avec l'aide d'experts humains.
  - Si les connaissances acquises sont jugées insignifiantes, inintéressantes, non pertinentes ou déviantes, cela peut indiquer la nécessité de révisions à des stades précoces tels que la formulation et la représentation du problème.
  - Il existe des problèmes pratiques connus dans de nombreuses méthodes d'apprentissage tels que le surapprentissage, les minima locaux ou la malédiction de la dimensionnalité qui sont dus à l'insuffisance des données, au bruit ou à des attributs non pertinents dans les données, à la nature d'une stratégie de recherche ou à une théorie de domaine incorrecte.

- **Utilisation de la base de connaissances (Fielding the knowledge base)**
  - Cette étape implique que les connaissances acquises soient utilisées. Les connaissances pourraient être intégrées dans un système de développement logiciel ou un produit logiciel, ou utilisées sans les intégrer dans un système informatique.

IA pour GL

- **Classification des travaux existants IA pour GL**
  - Utilisation des types d'activité comme ligne directrice pour regrouper les applications des méthodes ML dans les tâches SE.
  - Sept types d'activités, dont chacune contient des applications dans un certain nombre de tâches SE différentes.

Summary of existing ML applications in SE tasks

Activity type	SE task	ML method[1]
Prediction	Software quality (high-risk, or fault-prone component identification)	GP (Evett et al., 1998), NN (Hong and Wu, 1997; Khoshgoftaar et al., 1995, 1997; Lanubile and G. Visaggio, 1997), CBR (El Emam et al., 2001; Ganesan et al., 2000) DT (Briand et al., 1993; Porter and Silbey, 1990), CL (de Almeida and Matwin, 1999), ILP (Cohen and Devanbu, 1997)
	Software size	{NN, GP} (Dolado, 2000)
	Software development cost	DT (Briand et al., 1992), CBR (Briand et al., 1999), BL (Chulani et al., 1999)
	Project/software (development) effort	CBR (Shepperd and Schofield, 1997; Vicinanza et al., 1990), {DT, NN} (Srinivasan and Fisher, 1995), GA + NN (Shukla, 2000), {NN, CBR} (Finnie et al., 1997)
	Maintenance task effort	{NN, DT} (Jorgensen, 1995)
	Software resource analysis	DT (Selby and Porter, 1988)
	Correction cost	{DT, ILP} (de Almeida et al., 1998)
	Software reliability	NN (Karunanithi et al., 1992)
	Defects	BL (Fenton and Neil, 1999)
	Reusability	DT (Mao et al., 1998)
	Software release timing	NN (Dohi et al., 1999)
	Testability of program modules	NN (Khoshgoftaar et al., 2000)
Property/model discovery	Program invariants	ILP (Bratko and Grobelnik, 1993)
	Identifying objects in programs	NN (Abd-El-Hafiz, 2000)
	Process models	NN (Cook and Wolf, 1998), EBL (Garg and Bhansali, 1992)
Transformation	Transform serial programs to parallel ones	GP (Ryan and Ivan, 1999; Ryan, 2000)
	Improve software modularity	CBR + NN (Schwanke and Hanson, 1994)
	Mapping OO applications to heterogeneous distributed environments	GA (Choi and Wu, 1998)

[Zhang and Tsai]

Activity type	SE task	ML method
Generation and synthesis	Test cases/data	ILP (Bergadano and Gunetti, 1996), GA (Michael and McGraw, 1998; Michael et al., 2001)
	Software agents	GP (Qureshi, 1996)
	Design repair knowledge	CBR + EBL (Bailin et al., 1991)
	Design Schemas	IBL (Harandi and Lee, 1991)
	Data structures	GP (Langdon, 1995)
	Program/scripts	IBL (Bansali and Harandi, 1993), {CL, AL} (Minton and Wolfe, 1994)
	Project management schedule	GA (Chang et al., 2001)
Reuse library construction and maintenance	Similarity computing	CBR (Ostertag et al., 1992)
	Active browsing	IBL (Drummond et al., 2000)
	Cost of rework	DT (Basili et al., 1997)
	Knowledge representation	CBR (Fouque and Matwin, 1992)
	Locate and adopt software to specifications	CBR (Katalagarianos and Vassiliou, 1995)
	Generalizing program abstractions	EBL (Hill, 1987)
	Clustering of components	GA (Lee et al., 1998)
Acquisition or recovery of specifications	Derivation of specifications of system goals and requirements	CL (Van Lamsweerde and Willemet, 1998)
	Extract specifications from software	ILP (Cohen, 1995)
	Acquire knowledge for specification refinement and augmentation	{DT, NN} (Partridge et al., 2001)
	Acquire and maintain specification consistent with scenarios	EBL (Hall, 1995; 1998)
Development knowledge management	Collect and manage software development knowledge	CBR (Henninger, 1997)
	Capture and reuse design knowledge	CBR (Leake and Wilson, 2001)

[1] An explanation on the notations: {...} is used to indicate that multiple ML methods are each independently applied for the same SE task, and "... + ..." is used to indicate that multiple ML methods are collectively applied to an SE task.

[Zhang and Tsai]

Prédiction et estimation

- **Dans ce groupe, les méthodes ML sont utilisées pour prédire ou estimer :**
  - (1) software quality,
  - (2) software size,
  - (3) software development cost,
  - (4) project or software effort,
  - (5) maintenance task effort,
  - (6) software resource,
  - (7) correction cost,
  - (8) software reliability,
  - (9) software defect,
  - (10) reusability,
  - (11) software release timing, and
  - (12) testability of program modules.

Prédiction et estimation

- **Software quality prediction.**

  - GP a été utilisé pour générer des modèles de qualité logicielle qui prennent en entrée des métriques logicielles collectées plus tôt dans le développement et prédisent pour chaque module le nombre de défauts qui seront découverts plus tard dans le développement ou pendant les opérations. Ces prédictions serviront ensuite de base pour classer les modules, permettant ainsi à un gestionnaire de sélectionner autant de modules en haut de la liste que les ressources le permettent pour l'amélioration de la fiabilité.

  - Des modèles basés sur NN ont été utilisés pour prédire les défauts et les mesures de qualité logicielle.

  - Le CBR est appliqué à la modélisation de la qualité logicielle d'une famille de systèmes logiciels industriels à grande échelle et la précision est considérée comme meilleure qu'un modèle de régression linéaire multiple correspondant pour prédire le nombre de défauts de conception.

  - Une approche basée sur la DT est utilisée pour générer des modèles basés sur la mesure des composants à haut risque. La méthode proposée s'appuie sur des données historiques (métriques de versions ou de projets précédents) pour identifier les composants des propriétés sujettes aux pannes. Une autre approche basée sur la DT est utilisée pour construire des modèles de prédiction des composants Ada à haut risque

Prédiction et estimation

- **Software size estimation.**
  - NN et GP ont été utilisés pour valider la méthode basée sur les composants pour l'estimation de la taille du logiciel.

- **Software cost prediction.**
  - Des techniques basée sur la DT et CRB ont été proposées pour analyser les données d'ingénierie logicielle, et s'avère être une technique efficace pour l'estimation des coûts logiciels.

- **Software (project) development effort prediction.**
  - Les techniques IBL sont utilisées dans pour prédire l'effort de projet logiciel pour de nouveaux projets.
  - Des résultats empiriques obtenus (à partir de neuf ensembles de données industrielles différents totalisant 275 projets) indiquent que le CBR offre un complément viable aux techniques de prédiction et d'estimation existantes.
  - DT et NN sont été utilisés pour aider à prédire l'effort de développement logiciel. Les résultats étaient compétitifs avec les méthodes conventionnelles telles que COCOMO et les points de fonction. Le principal avantage des systèmes d'estimation basés sur DT et NN est qu'ils sont adaptables et non paramétriques.

- **Maintenance task effort prediction.**
  - Des modèles sont générés en se basant sur NN et DT, et de méthodes de régression, pour la prédiction de l'effort des tâches de maintenance logicielle dans. L'étude mesure et compare la précision des prédictions pour chaque modèle, et conclut que les modèles basés sur la DT et sur la régression multiple ont de meilleurs résultats de précision. Il est recommandé d'utiliser des modèles de prédiction comme instruments pour étayer les estimations d'experts et analyser l'impact des variables de maintenance sur le processus et le produit de la maintenance.

- **Software resource analysis.**
  - DT est utilisé dans l'analyse des données de ressources logicielles pour identifier les classes de modules logiciels qui ont un effort de développement élevé ou des défauts (le concept de « élevé » est défini par rapport au quartile supérieur par rapport aux données passées). Seize systèmes logiciels sont utilisés dans l'étude. Les arbres de décision identifient correctement 79,3 pour cent des modules logiciels qui ont fait l'objet d'un effort de développement élevé ou de défauts.

Prédiction et estimation

- **Correction cost estimation.**
  - Une étude empirique a été réalisée où DT et ILP sont utilisés pour générer des modèles d'estimation des coûts de correction dans la maintenance logicielle. Les modèles générés s'avèrent précieux pour aider à optimiser les allocations de ressources dans les activités de maintenance corrective et à prendre des décisions concernant le moment de restructurer ou de réorganiser un composant afin de le rendre plus maintenable. Une comparaison conduit à une observation que les résultats basés sur ILP fonctionnent mieux que les résultats basés sur DT.

- **Software reliability prediction.**
  - Les modèles de croissance de la fiabilité des logiciels peuvent être utilisés pour caractériser la façon dont la fiabilité des logiciels varie avec le temps et d'autres facteurs. Les modèles offrent des mécanismes pour estimer les mesures de fiabilité actuelles et pour prédire leurs valeurs futures.
  - Certaines travaux rapportent l'utilisation de NN pour la prévision de la croissance de la fiabilité des logiciels.
  - Une comparaison empirique est menée entre les modèles basés sur NN et cinq modèles de croissance de la fiabilité des logiciels bien connus en utilisant des ensembles de données réelles provenant d'un certain nombre de projets logiciels différents. Les résultats indiquent que les modèles basés sur NN s'adaptent bien à différents ensembles de données et ont une meilleure précision de prédiction.

- **Defect prediction.**
  - BL a été utilisé pour prédire les défauts logiciels et montre le potentiel des réseaux de croyances bayésiennes (BBN) en intégrant plusieurs perspectives sur la prédiction des défauts dans un modèle unique et unifié.
  - Les variables du système BBN ont été choisies pour représenter les processus du cycle de vie de la spécification, de la conception et de la mise en œuvre, et des tests (Problème-Complexité, Conception-Effort, Conception-Taille, Défauts-Introduit, Test-effort, défauts détectés, densité de défauts au test, nombre de défauts résiduels et densité de défauts résiduels).
  - Les relations causales appropriées entre ces processus du cycle de vie du logiciel sont ensuite capturées et reflétées sous forme d'arcs reliant les variables.
  - Un outil est ensuite utilisé en ce qui concerne le modèle BBN de la manière suivante. Pour des faits donnés sur Design-Effort et Design-Size comme entrée, l'outil utilisera l'inférence bayésienne pour dériver les distributions de probabilité pour les défauts introduits, les défauts détectés et la densité de défauts.

Prédiction et estimation

- **Reusability prediction.**
  - Des modèles prédictifs ont été proposés via DT dans pour vérifier l'impact de certaines propriétés internes des applications orientées objet sur la réutilisabilité.
  - L'effort est concentré sur l'établissement d'une corrélation entre la réutilisabilité des composants et trois attributs logiciels (héritage, couplage et complexité).
  - Les résultats expérimentaux montrent que certaines métriques logicielles peuvent être utilisées pour prédire, avec un haut niveau de précision, les classes potentiellement réutilisables.

- **Software release timing.** Comment déterminer le calendrier de publication du logiciel est une question qui a un impact à la fois sur le développeur du produit logiciel et sur l'utilisateur et le marché. Une méthode, basée sur NN, a été proposée pour estimer le moment optimal de sortie du logiciel. La méthode adopte le critère de minimisation des coûts et le traduit en un problème de prévision de séries temporelles. NN est ensuite utilisé pour estimer le temps de détection de défaut dans le futur.

- **Testability prediction.** NN a été utilisée pour prédire la testabilité de modules logiciels à partir de mesures statiques du code source. L'objectif de l'étude est de prédire une quantité comprise entre zéro et un dont la distribution est fortement asymétrique vers zéro, ce qui s'avère difficile pour les techniques statistiques classiques. Les résultats font écho à la caractéristique principale des modèles prédictifs basés sur NN qui ont été discutés jusqu'à présent : sa capacité à modéliser des relations non linéaires.

Découverte de propriétés et de modèles

- Les méthodes ML sont utilisées pour identifier ou découvrir des informations utiles sur les entités logicielles.
  - ILP a été utilisée pour découvrir les invariants de boucle.
  - NN est utilisé pour identifier des objets dans des programmes procéduraux dans le but de faciliter de nombreuses activités de maintenance (réutilisation, compréhension).
    - L'approche est basée sur l'analyse de cluster et est capable d'identifier des types de données abstraits et des groupes de routines qui font référence à un ensemble commun de données.

Transformation

- Un système GP a été proposé pour transformer des programmes série en programmes parallèles fonctionnellement identiques. La propriété fonctionnelle identique entre l'entrée et la sortie de la transformation peut être prouvée, ce qui améliore considérablement les opportunités d'utilisation du système dans des environnements commerciaux.
- Un assistant d'architecture de module a été proposé pour aider les architectes logiciels à améliorer la modularité des grands programmes. Un modèle de modularisation est établi en termes de regroupement et de classification des voisins les plus proches, et est utilisé pour faire des recommandations pour réorganiser l'appartenance au module afin d'améliorer la modularité. L'outil apprend des jugements de similarité qui correspondent à ceux de l'architecte logiciel en effectuant une rétro-propagation sur un réseau de neurones spécialisé.
- GA est utilisé pour expérimenter et évaluer un modèle de partitionnement et d'allocation pour mapper des applications orientées objet vers des environnements distribués hétérogènes. En distribuant efficacement les composants logiciels d'un système orienté objet dans un environnement distribué pour atteindre des objectifs de performances tels que l'équilibrage de charge, la maximisation de la simultanéité et la minimisation des coûts de communication.

Génération et synthèse

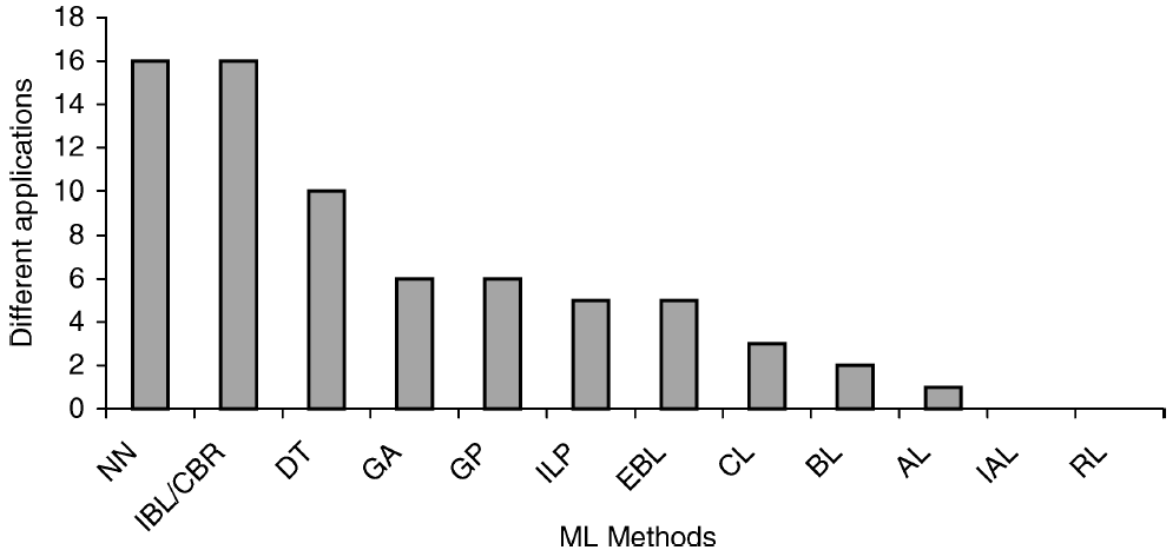
- Une méthode de génération de cas de test est proposée qui est basée sur ILP. Un ensemble de tests adéquat est généré à la suite d'un apprentissage inductif de programmes à partir d'ensembles finis d'exemples d'entrées-sorties. La méthode évolue bien lorsque la taille ou la complexité du programme à tester augmente. Il cesse d'être pratique si le nombre d'alternatives (ou d'erreurs possibles) devient trop important.

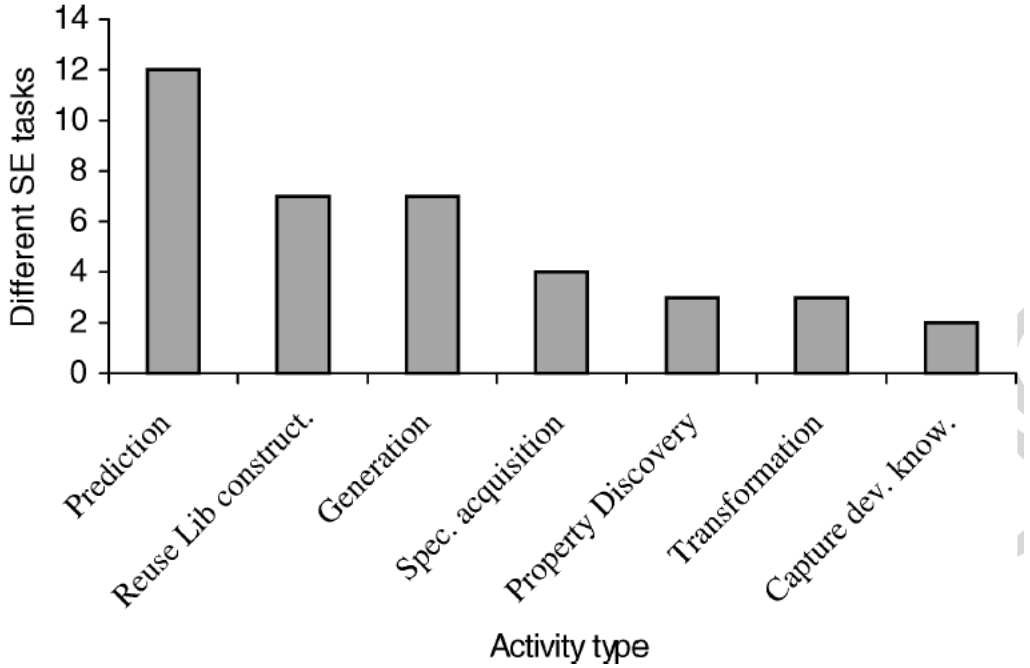
- GA a été utilisée pour générer des données de test dynamiques pour les programmes C/C++. L'outil est entièrement automatique et prend en charge toutes les constructions du langage C/C++. Des résultats de test ont été obtenus pour des programmes contenant jusqu'à 2000 lignes de code source avec des conditions complexes imbriquées. La synthèse de scripts shell Unix à partir d'une spécification de haut niveau est rendue possible par IBL. L'outil dispose d'un mécanisme de récupération qui permet à une source analogique appropriée d'être récupérée automatiquement en fonction d'une description d'un problème cible.

- Un prototype d'environnement de génie logiciel combine CBR et EBL pour synthétiser les règles de réparation de conception pour la conception de logiciels.

- CL et AL sont utilisés dans la synthèse de programmes de recherche pour un générateur de code Lisp dans le domaine des problèmes combinatoires de satisfaction de contraintes d'entiers.

- GA est à l'origine de l'effort de génération de calendriers de gestion de projet. En utilisant une fonction d'objectif programmable, la technique peut générer une allocation presque optimale des ressources et un calendrier qui satisfont une structure de tâche et un pool de ressources donnés.





[Zhang and Tsai]

	NN	IBL CBR	DT	GA	GP	ILP	EBL	CL	BL	AL	IAL	RL
Prediction	√	√	√	√	√	√		√	√			
Discovery	√					√	√					
Transformation	√	√		√	√							
Generation		√		√	√	√	√	√		√		
Reuse		√	√	√			√					
Acquisition	√		√			√	√	√				
Management		√										

Guideline for selecting AL/EBL

Domain knowledge	Correct and complete domain knowledge required.
Training data	Small number of training cases needed.
Target function	Form of learned function is known and expressed as set of rules.
Advantages	Ability to use domain theory and deductive reasoning to generalize from small amount of training data.
Disadvantages	Can be misled if domain theory is not correct or complete.
Applicable SE activities	Requirement engineering (generalizing specifications, scenario based specification acquisition).

Guideline for selecting BL

Domain knowledge	Required in terms of prior probabilities.
Training data	Incrementally decrease or increase the estimated probability.
Target function	Can be represented either explicitly (as a MAP hypothesis) or implicitly (as providing classifications for unseen cases).
Advantages	Flexible in learning target function. Probabilistic prediction.
Disadvantages	Requiring many initial probabilities. Computational cost.
Applicable SE activities	Predicting defects. Analyzing cost models.

Guideline for selecting CL

Domain knowledge	Not required.
Training data	Need to be correct and free of errors. Organized as positive and negative examples.
Target function	Form of learned function is known and expressed in terms of a conjunction of constraints.
Advantages	Version Space (general-to-specific ordering) and Candidate_Elimination algorithm provide a general structure for concept learning problems.
Disadvantages	Learning algorithm not robust to noisy data. H may be incomplete due to its inductive bias.
Applicable SE activities	Deriving system specifications and properties.

Guideline for selecting DT learning

Domain knowledge	Not required.
Training data	Adequate data needed to avoid overfitting. Missing values tolerated.
Target function	Discrete-valued. Form of learned function is known and expressed as decision trees.
Advantages	Robust to noisy data. Capable of learning disjunctive expressions.
Disadvantages	Overfitting.
Applicable SE activities	Predicting faults, costs, development efforts, and reusability. Acquiring specifications. Diagnosing system errors.

Guideline for selecting GA/GP learning

Domain knowledge	Not required.
Training data	Not needed (some test data may be needed for fitness evaluation).
Target function	Expressed as bit strings (GA) or program trees (GP).
Advantages	Suited to tasks where functions to be approximated are complex. Algorithms can be easily parallelized.
Disadvantages	Crowding. Bloating.
Applicable SE activities	Predicting faults, size, and development efforts. Program transformation. Program synthesizing. Prototyping.

Guideline for selecting IBL/CBR

Domain knowledge	Not required.
Training data	Plenty data needed.
Target function	Can be represented either explicitly (as a linear function) or implicitly (as providing classifications for unseen cases). Local approximation.
Advantages	Training is fast. Can learn complex functions. Do not lose information.
Disadvantages	Slow at query time. Curse of dimensionality.
Applicable SE activities	Reuse library. Tasks scheduling and planning. Diagnosis.

[Zhang and Tsai]

Guideline for selecting ILP learning

Domain knowledge	Background (domain) knowledge required.
Training data	Organized as positive and negative examples.
Target function	Form of learned function is known and expressed as set of rules.
Advantages	Expressive and human readable representation of learned function. Induction formulated as inverse of deduction. Background knowledge guided search.
Disadvantages	Not robust to noisy data. Intractable search space in general case. Increased background knowledge results in increased complexity of hypothesis space. No guarantee to find the smallest or best set of rules.
Applicable SE activities	Predicting faults, and cost. Identifying program invariants. Test data generation. Reverse engineering (extracting specifications from software).

Guideline for selecting IAL

Domain knowledge	Required but does not have to be perfect.
Training data	Training data needed and possibly containing errors.
Target function	Form of learned function can be either unknown or known, depending on how domain theory and training data are combined to constrain the search process.
Advantages	Enjoying the benefits of both inductive and analytical learning. Better generalization accuracy (due to domain knowledge). Ability to circumvent imperfect domain knowledge (via training data). A framework where different inductive and analytical methods can be accommodated.
Disadvantages	Complex.
Applicable SE activities	Predicting faults, size, development efforts, reliability and testability. Generating test cases, and specifications. Identifying objects and module properties. Recognizing patterns in software systems.

Guideline for selecting NN learning

Domain knowledge	Not required.
Training data	Need to have plentiful data. Training data represented as many attribute-value pairs and possibly containing errors.
Target function	Form of learned function is unknown but human readability of learned result is unimportant.
Advantages	Robust to errors in training data. Can learn complex functions (non-liner, continuous functions). Parallel, distributed learning process.
Disadvantages	Slow training and convergence process. Multiple local minima in error surface. Overfitting.
Applicable SE activities	Predicting faults, size, development efforts, reliability and testability. Identifying objects and module properties. Recognizing patterns in software systems.

Guideline for selecting RL

Domain knowledge	Not required.
Training data	Not available.
Target function	Learned function is represented as a control strategy from state space to action space.
Advantages	An unsupervised paradigm geared toward goal-directed learning and decision-making by an agent through direct interaction with its environment without relying on training data or domain theory.
Disadvantages	Exploration-exploitation dilemma.
Applicable SE activities	Scheduling, planning and resource allocation in software project management.

[Zhang and Tsai]

Microservice Identification Using a Multi-Objective Genetic Algorithm

Motivation : Microservices

- Moving to the cloud is a natural evolution for :
  - Making use of a powerful computing environment,
  - Providing customers with fully hosted, efficient, reliable as well as scalable software.
- Adopting native Cloud architectures such as microservices become unavoidable
- Instead of building an application as a single unit (monolithic architecture), microservice architectural style considers a suite of :
  - Small
  - independently deployable services
  - “focused on doing one thing well”
  - that communicate with each other via APIs.
- Such organisation eases software comprehension by structuring the services around well-defined functionalities allowing, among other benefits, to reduce maintenance effort and time.
- Another important characteristic of microservices is that they are not limited by the “one-size-fits-all” factor in terms of programming languages and data storage technologies - i.e., the technology stack for each microservice within the architecture could be different depending on the needs

Motivation: Migrating to Microservices

- Any refactoring procedure from monolithic to microservice-based architecture usually starts by the ”decomposition phase” also referenced to as ”microservice identification phase”.
- It consists of dividing a system into smaller parts according to microservices’ specificities.
- Problems of Monolithic architecture :
  - Developers have to resolve the dependencies with existing services when adding or modifying functionalities and redeploy the whole application.
  - Furthermore, a monolith behaves very badly when it comes to failures. In fact, the failure of a service may break down the entire application.
- To address these limitations, a monolithic application can be re-architectured into a microservice-based one.
  - The self-contained nature of microservices allows them to be changed, deployed and scaled separately.
  - Therefore, the system is able to “limit the scope of a failure” to specific microservices and set up recovery mechanisms for a better user experience.

GOAL

- Propose an approach ensuring that the identified microservices are the best possible ones.
- The intuitive way is to apply an algorithm to all the possible partitions of the system.
  - However, it’s a non trivial process which requires an exponential number of operations as the size of the system to be decomposed increases making an exhaustive search for the optimal solution impossible.
  - Therefore, we introduced in our approach some widely used meta- heuristics in Search Based Software Engineering: genetic algorithms.
  - We were particularly interested in multi-objective genetic algorithms that present more realistic formulations to our optimization problem.

GOAL

- Finding the best partitioning of classes into cohesive micro services is not an obvious task for developers as the number of possible partitions can be very large causing a combinatorial explosion.
- The search space tends to be enormous as the number of possible partitions is given by:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

  - B_n counts the number of different possibilities of how a given set of n classes can be divided into microservices.
    - For instance, consider the application X which contains 87 classes in the version 20xx. To find the right microservice partitioning for application X, the number of combinations of its 87 classes, a developer needs to explore $B_{n+1} \approx 6.39 \times 10^{98}$ possible ways to decompose the system.
    - Due to this huge search space, an exhaustive search is unsuitable. Instead, we based our approach on Genetic Algorithm (GAs).

GOAL

- Genetic algorithms (GAs) take after the evolutionist theory explaining the origin of species.
- They assimilate the search-space to a population and mimic its evolution system through genetic operators such as selection, crossover and mutation.
- Each individual represents a possible decomposition of the system (a solution) and is referred to as a chromosome.
- Chromosomes are made of elements holding their characteristics called genes.

Problem formulation

- The target model entities (i.e., microservices) are identified as partitions of OO classes of the source model.

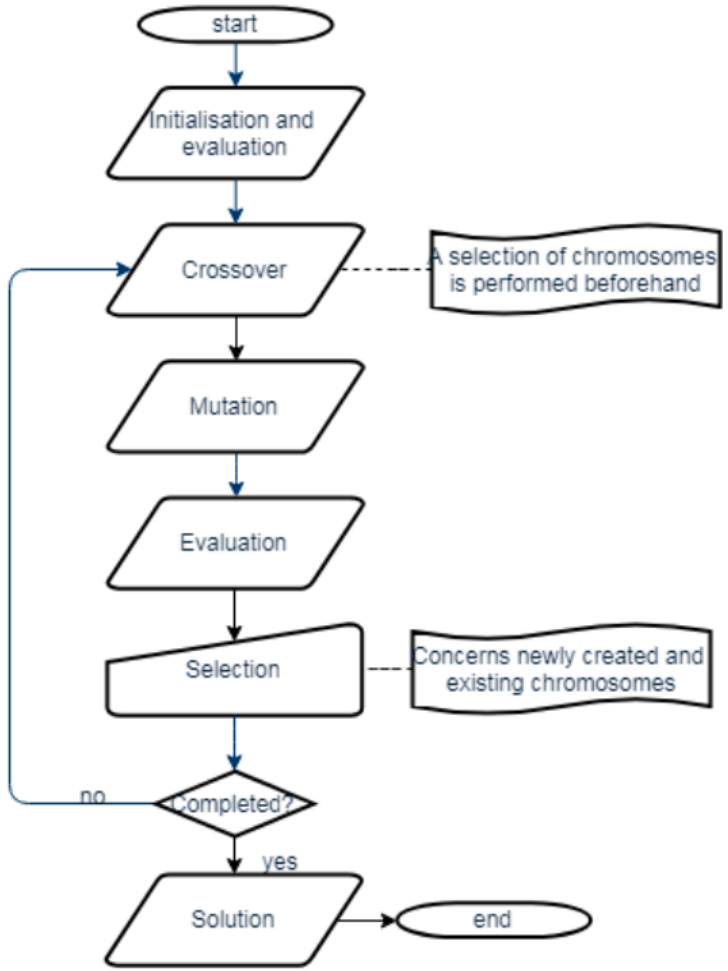
- Partition need to be based on degree of coupling of micro services, their cohesion and their data autonomy
  - Data Autonomy
    - The shared database is a common integration mode in monolithic applications where multiple services usually manipulate data stored in a single database.
    - In the context of microservice architectures, the shared data integration is not favored as it has a direct impact on the autonomy of the entities.
    - For a microservice to be thoroughly independent, it should have an autonomous data management that suppress the need for any external data.
    - We use the data autonomy characteristic in the main line of our identification process.

Problem representation

- To implement a biological evolution, a genetic algorithm should define several elements:
  - The encoding of the problem
  - The fitness function
  - The genetic operators
  - The initial population

Problem representation

- The first step in a generic GA is to define and evaluate an initial population according to a fitness function that measures the chromosomes' adaptation to the objectives of the optimization problem.
- The adaptation metric conditions a solution's existence in the process.
- Naturally, chromosomes with better attributes will have more chance to carry on their genes to the next generation established in the second phase.
- GAs perform selection and crossover operations on the population to create new chromosomes.
- The mutation operator is then applied (on newly created chromosomes) to avoid an early convergence to any solution before exploring the entire space.
- Lastly, a stopping condition has to be specified for these metaheuristics to wrap the exploration.



Problem representation

- Encoding of the Problem
  - A potential solution in our approach is a partition of existing classes in such a way that
    - none of its subsets is empty
    - their union represents all the classes of the OO application and
    - their intersection is empty.
  - In traditional GAs, a solution is coded as a single chromosome consisting of one or more gene(s).
  - In our case, the genes are represented by sets of classes.

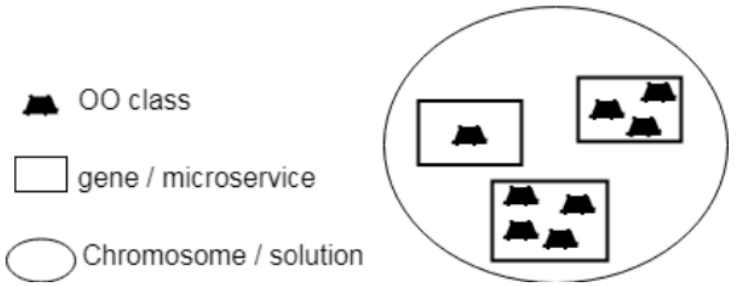


Figure 2: Encoding of a chromosome

Problem representation

- Fitness Function
  - The evaluation is a key element of each search-based technique.
  - It guides the process by apposing a tag on the elements within the search-space, enabling the recognition of worthwhile solutions.
  - In the system decomposition context, the microservices constituting the target architecture are identified among a set of candidate ones according to their response to a well-defined fitness function, which is a weighted aggregation of two sub-functions.
    - The first sub-function measures the quality based on the structural relationships between source code entities. T
    - he second one evaluates the quality of microservices relying on the data shared between these entities.

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- A microservice is viewed as a set of classes collaborating to provide a given function.
- This collaboration can be determined from source code through
  - The internal coupling measure, that represents the degree of direct and indirect dependencies between the set of classes. The more two classes of a candidate microservice use each other's methods, the more they are internally coupled.
  - Furthermore, the collaboration can be determined by measuring the number of volatile data (i.e., not persistent data) such as attributes whose use is shared by these classes. It reflects the internal cohesion measure.
- The internal coupling (ICoupling) and cohesion (ICohesion) measures are grouped under a function (F_{sem}) evaluating the semantic aspect of the functionality provided by a microservice M.

$$F_{sem}(M) = \frac{1}{2}\left(ICoupling(M) + ICohesion(M)\right) \qquad (1)$$

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- Microservices are self-sufficient entities which can be developed, upgraded, deployed, and scaled independently from each other.
- Therefore, in order that a set of classes represents a microservice, their dependencies on external classes should be minimal.
- This can be measured using external coupling(ECoupling), which evaluates the degree of direct and indirect dependencies between the classes belonging to the microservice and the external classes. To express the structural and behavioral autonomy of a microservice M we defined F_{auto} as follows:

$$F_{auto}(M) = ECoupling(M) \tag{2}$$

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- The two previous functions are aggregated in one that measures the functional modularity of a microservice M.
- Its purpose is to maximise the internal coupling and cohesion and minimise the external coupling.

$$F_{func}(M) = \frac{\alpha F_{sem}(M) - \beta F_{auto}(M)}{\alpha + \beta} \qquad (3)$$

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- **Internal Coupling**
  - The internal coupling evaluates the dependencies between the classes of a microservice M by measuring the frequency of internal method calls.
  - It is given by the following expression:

$$ICoupling\,(M) = \frac{\sum Coupling\,(C_i, C_j) - SumSDev}{nbPossiblePairs} \qquad (5)$$

$$Coupling\,(C_i, C_j) = \frac{nbCalls\,(C_i, C_j) + nbCalls\,(C_j, C_i)}{nbTotalCalls} \qquad (6)$$

    - Where
      - (C_i,C_j) is a pair of classes of M
      - nbPossiblePairs is the number of possible pairs of classes in M,
      - n is the number of classes in M,
      - i = 1, ..., n and j = 1, . . . , n.
      - SumDev is the sum of the standard deviations between the Coupling values of all the possible pairs in M.
      - nbCalls(C_i,C_j) the number of method calls fromC_i to C_j
      - nbTotalCalls : the total number of method calls in the OO application.
      - We introduced the standard deviation to ensure that very big or very small coupling values between a small sub-set of microservice classes do not impact the overall evaluation of the internal coupling.

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- External Coupling
  - unlike the internal coupling that operates between classes of the same mi- croservice, the external coupling evaluates the dependencies between microservices.
  - Hence, the coupling measure is performed on pairs of classes associated to different microservices.
  - The lesser a microservice solicits external classes, the smaller is the value of the metric, the better it is.
    - k = 1, . . . , m with m the number of classes in the application minus the number of classes in M and $C_k \notin M$.

$$ECoupling\,(M) = \frac{\sum Coupling\,(C_i, C_k) - SumSDev}{nbPossiblePairs} \qquad (7)$$

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- Internal Cohesion
  - Internal cohesion evaluates the strength of interactions between classes.
  - Generally, if the methods of two classes manipulate the same attributes, these classes are more interactive.
  - The metric used to evaluate the internal cohesion is a variation of the Loose Class Cohesion (LCC) metric.
    - It is the ratio between the number of direct (nbDC) and indirect (nbIC) connections between the methods of a microservice M to the number of possible connections(nbPossibleC).

$$ICohesion\,(M) = \frac{nbDC\,(M) + nbIC\,(M)}{nbPossibleC} \qquad (8)$$

Fitness Function : Measuring the Quality of a Microservice Based on Structural and Behavioral Dependencies

- Two methods are directly connected if they are directly connected to an attribute (i.e., they both have access to the attribute).
- A method $m1$ is indirectly connected to an attribute j when there is a method $m2$ directly connected to both $m1$ and j.
- Note that the metric we implemented does not include the connections between the methods of the same class so that its value reflects the real collaboration level between the classes of a microservice.

Fitness Function :Measuring the Quality of a Microservice Based on Data Autonomy

- A microservice can be completely data autonomous if it does not require any data from others.
- In order to achieve a certain degree of autonomy, a microservice should maximize the internal data (IData) manipulations between its classes, while minimizing the accesses to external data (EData). we defined F_{data} to measure that characteristic.

$$F_{data}(M) = \frac{\lambda IData(M) - \gamma EData(M)}{\lambda + \gamma} \qquad (4)$$

  - λ and γ as well as α and β are weights that can vary accordingly.
  - F (M) means the application of both fitness functions F_{func} (M) and F_{data} (M).

Fitness Function :Measuring the Quality of a Microservice Based on Data Autonomy

- Internal and External Data
  - The internal and external data in a microservice M are evaluated through data access.
  - IData represents the ratio of the data only manipulated by the classes in M to the total amount of data manipulated in M.
  - EData measures the dependency of other microservices on M .
  - It is the ration of data manipulated by classes other than those in M to the total amount of data manipulated in M.

Problem representation/modeling : Genetic Operators

- Genetic operators explore the search space by creating new solutions from existing ones.
- Their role is to allow the evolution of the population until an acceptable solution is reached.
- They are essentially of three types:
  - Selection
  - Crossover
  - Mutation.

Problem representation/modeling : Genetic Operators

- Selection operators
  - There are two selection operators used in GAs.
    - The first one intervenes in the crossover operation by selecting chromosomes called parents.
    - The second operator is in charge of the constitution of the next generation.
  - To the question ”On which basis is this selection made?”, several answers have been provided.
    - One of them is the ranking technique based on the “strongest law” and stipulates that only the fittest chromosomes have to be part of future generations.
      - However, this can cause a decrease in genetic diversity because weak chromosomes won’t be able to spread their genes.
    - Random is another selection method that selects chromosomes randomly from a population.
      - But still, there is a risk for the generated population to be of poor composition.
  - The method applied in this approach is the roulette wheel that brings a balance between the previous two techniques.
    - It highlights the best chromosomes, while giving a chance to the less good ones to be selected by assigning proportional probabilities to their adaptation before the selection.
    - The wheel roulette was slightly modified for the selection of chromosomes of the new generation.
    - An aging process that determines the life span of chromosomes was integrated. This in view of a certain balance in the growth of the population.

Problem representation/modeling : Genetic Operators

- Crossover operators
  - From two parents, the crossover operators generate two new chromosomes each inheriting portions of their parents' genes.
  - We used the following algorithm : The algorithm proceeds by keeping all the genes of one of the parents, adding a part of the genes of the second parent and then eliminating duplicates.
  - With this, we ensure that new chromosomes contain all of the existing application's classes and guarantee that a class exists in one and only one microservice.

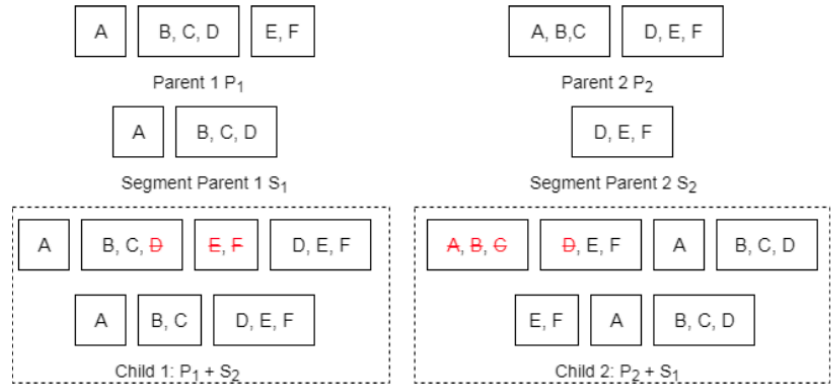


Figure 3: Chardigny's crossover operation

Problem representation/modeling : Genetic Operators

- Mutation operators
  - they introduce new genes in the chromosomes.
  - In the original version of GAs, the solution was coded in binary, so the mutation operation consists of randomly inverting a bit.
  - In our case, since losing any gene is not acceptable, the mutation will be either a fusion or a separation.
    - While a separation operation scatters a gene's elements into distinct genes, the fusion gathers multiple genes into one.
    - The mutation comes with a predefined probability that is usually very low to minimize the changes.

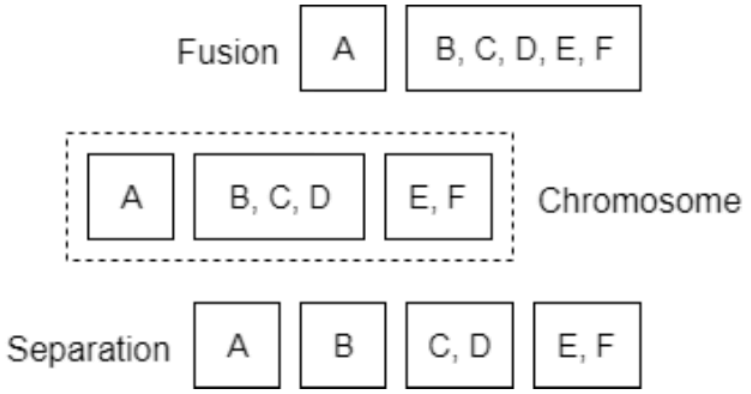


Figure 4: Mutation forms

Problem representation/modeling : Initial Population

- Initial Population
  - The definition of the initial population is a crucial phase that can contribute to the success or failure of a GA.
  - The more this initial population is diversified, the higher is the rate of the emergence of interesting species.
  - The exploration guided by the fitness function will then be likely to produce more accurate solutions.

Multi-Objective GA

- There are essentially two modeling patterns in optimization:
  - single-objective
  - multi-objective [15].
- The difference between them lies in the evaluation technique used on the fitness function.
  - single-objective optimization : all the objectives identified are combined into a single one i.e. solutions are compared following the evaluation of a sole final objective.
  - Multi-objective : provides solutions with the best trade-off in respect with the objectives.
    - Each objective in the fitness function is evaluated separately and optimal solutions are those where no objective can be improved without degrading the value of one or more of the other objectives.

Algorithm 1: Multi-objective GA

input : Set of OO classes, Iterations I, Mutation probability M_p, Elite rate E_r

output: Approximate Pareto set *pareto*

Initialize Population P;
for $i \leftarrow 1$ **to** I **do**
    foreach *chromosome* $c \in P$ **do**
        Calculate $F_{func}(c)$ and $F_{data}(c)$;
    end
    Update *pareto*;
    $E_1 = \text{Elite}(F_{func})$;
    $E_2 = \text{Elite}(F_{data})$;
    for $i \leftarrow 1$ **to** $N_{selections}$ **do**
        foreach *chromosome* $c \in P$ **do**
            Generate weights and Calculate $F(c)$;
        end
        $(C_1, C_2) =$ Select 2 chromosomes;
        $(offspring_1, offspring_2) = \text{Crossover}(C_1, C_2)$;
        $\text{Mutation}(offspring_1, M_p)$;
        $\text{Mutation}(offspring_2, M_p)$;
        $P \leftarrow offspring_1 + offspring_2$;
    end
    $P_{i+1} \leftarrow E_1 + E_2 + E_r * Selection(pareto)$;
    $P_{i+1} \leftarrow Selection(P)$;
end

Experimentation and validation

- To validate our approach, we conducted some experiments on a number of open source projects in the attempt to answer three research questions empirically.
- The first question concerns the introduction of a specific genetic algorithm in the process.
  - RQ1: does the proposed approach provide an efficient system decomposition?
  - RQ2: are the microservices identified without considering the data autonomy function relevant?
  - RQ3: is the quality of microservices identified including the data autonomy objective function better?

Experimentation and validation

- Experimental Protocol
  - We adopted the following protocol in the purpose of answering the previous research questions.
    - First, we used a microservice identification tool developed in the Java programming language to partition four OO open source projects based on two configurations.
    - Initially, all the information needed by our tool was computed from the source code. Then, the identifier was configured to extract an indicated number of microservices.
    - Based on a resulting architecture
      - we apprehend RQ1 by comparing its microservices to a manual system decomposition in terms of class distribution.
      - The same protocol was applied for RQ2 with a difference in the fitness function as the data autonomy was not involved.
      - Lastly, we used the results obtained from the previous questions and the performance of the fitness function in each test case to answer RQ3.
  - To compare our solution to the target system, we classify the microservices obtained manually in three categories:
    - Excellent microservice is considered excellent, any microservice matching perfectly the one identified by our approach.
    - Correct microservice the microservices that can be obtained by at most three composition/decomposition operations of the ones identified by our approach are considered as correct mi- croservices. The composition and decomposition operations of microservices make it possible to adjust their granularity to that chosen by the software architect. Indeed, the granularity of a microservice is known to be strongly dependent on the style of each architect.
    - Bad microservice the label bad goes to microservices that are neither excellent nor correct.

Experimentation and validation

- Data Collection
  - Among many existing projects, we decided on:
    - JPetstore, a small-sized online pet shop with a known (manual) microservice architecture
    - Spring- Blog, a clean-design blog system implemented with Spring Boot
    - JForum, a bulletin board and
    - Roller, a full-featured, multi-user and group-blog server suitable for blog sites.
  - The source code of all these applications can be found at https://github.com/wj86.

Table 1: Data collection

Application	Class	Model	SLOC
JPetstore	28	9	1722
SpringBlog	78	19	3557
JForum	334	37	31841
Roller	530	58	53325

Experimentation and validation

- Microservices Identification Results
  - A first batch of experimentation consists of identifying microservices in each application using the fitness function F.
  - Since F's purpose is to maximize both F_{func} and F_{data}, the results displayed in table 2 reflects the functional as well as the data autonomy of the solution.
  - we executed our program with an initial population of 50 diverse individuals in a neutral environment.
    - Meaning that the algorithm wasn't influenced by any preference in objectives.
  - The number of extracted microservices is specified along with the the projects' names. A
  - lso, it is noteworthy that in case of SpringBlog, JForum and Roller, our tool was set to extract respectively seven, nine and sixteen microservices unlike Jpetstore that was totally unaware of such parameter.

Table 2: Global Fitness Function

Metric	JPetstore(3)	SpringBlog(7)	JForum(9)	Roller(16)
F_{func}	0.78	0.93	0.56	0.63
F_{data}	0.4	0.2	0.09	0.0
$ICoupling$	0.89	0.75	0.69	0.85
$ICohesion$	1E-4	9E-2	0.16	0.0
$ECoupling$	1E-3	3.9E-4	2.1E-7	2.5E-6
$IData$	0.5	0.3	0.12	0.04
$EData$	0.15	0.41	0.87	0.89

Experimentation and validation

- The second experimentation considers F_{func} alone in the identification pro- cess.
- Therefore the microservices are evaluated on three criteria:
  - the internal coupling
  - the internal cohesion
  - the external coupling. T

Table 3: Functional Modularity

Metric	JPetstore(4)	SpringBlog(7)	JForum(9)	Roller(16)
$ICoupling$	0.87	0.5	0.76	0.85
$ICohesion$	7.3E-4	0.14	0.22	0.0
$ECoupling$	1E-3	8.1E-6	3.2E-7	1.1E-7

Results Interpretations

- With a detailed representation of each sub-function (2, 3) we will be able to interpret the functional modularity as well as the data autonomy of the microservices identified.

Table 4: Comparison approach - target system

Function	Category			Quality	
	Excellent	Correct	Bad	Precision	Recall
F	0	3	0	100%	100%
F_{func}	0	3	1	75%	100%

Experimentation and validation

- Research Questions Answers
  - To the question RQ1, we can confidently answer that the proposed microservice identification approach is efficient. The JPetstore use case shows that we were able to extract a fair amount of microservices according to the target sys- tem. In fact, with a precision and recall of 100%, the global fitness function was able to identify all the microservices identified manually.
  - We can make the same statement concerning RQ2 because even if the identification was less pre- cise we can see through table 3 that the microservices are highly coupled and autonomous as the external coupling values are on the minimal side.
  - For RQ3, although the case study directly indicates that the microservices are better using the global fitness function, we compared F and F_{func} on a metric level considering all the applications.
    - The digits visible in table 5 ex- press the number of applications in which one function outperforms another in the corresponding metric. It appears that F tends to identify more coupled microservices and F_{func} more autonomous ones. Also, the cohesion is better most of the time with F_{func}. From the analysis, we concluded that our researches could be extended to the causal factors of those behaviours.
    - A general observation from the results is that the fitness function tends to rely more on the functional modularity objective with the growing number of the classes in the application. How can we refine the fitness function in order to guide the search towards individuals even stronger in all of the objectives? Enhancing the data autonomy fitness function could be a starting point.

Table 5: Metric-level comparison

	ICoupling	ICohesion	ECoupling
F	2	0	1
F_{func}	1	3	2

End