Année 2024-2025

Rapport Technique de stage

Génération d'instances artificelles pour des problèmes d'optimisation complexes

Lien de l'archive :

https://drive.google.com/drive/folders/1daJmUFItEmPg45QqTxLabGDmN5PiKuwg?usp=sharing

Yastene Guendouz

Encadré par M. Veerapen et Mme. Kessaci

POLYTECH LILLE | Université de Lille

CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille

0) Pré-requis & organisation

- Python : 3.10.11
- Arborescence de travail (crée-la à la racine du projet) :

```
projet_itc/
├── data/
│   ├── real/
│   │   ├── real.csv
│   │   ├── real_no_erlangen.csv        # (généré, voir §2.0)
│   │   └── real_only_erlangen.csv      # (généré, voir §2.0)
│   ├── synthetic/
│   │   ├── synthetic_ctgan.csv         # (généré, §2.2)
│   │   ├── synthetic_tvae.csv          # (généré, §2.3)
│   │   └── ectt/                       # .ectt générés (§3)
│   └── itc2019/                        # (pour la suite, GA)
├── models/
│   ├── ctgan_model.pkl                 # (généré)
│   ├── tvae_model.pkl                  # (généré)
│   ├── ctgan_model_no_erlangen.pkl     # (optionnel)
│   └── tvae_model_no_erlangen.pkl      # (optionnel)
├── outputs/
│   ├── figures/
│   └── results/
├── scripts/
│   ├── auto_cluster_csv.py
│   ├── train_ctgan.py
│   ├── train_tvae.py
│   ├── csv2ectt.py
│   ├── run_solver_and_plot.py
│   └── show_umap_solved.py
├── requirements.txt
└── README.md
```

0.1) Installation des dépendances

pandas==2.2.3
numpy==1.26.4
scikit-learn==1.7.0
matplotlib==3.10.0
umap-learn==0.5.7
hdbscan==0.8.40
liac-arff==2.5.0
sdv==1.23.0
ctgan==0.11.0
torch==2.2.2
rdt==1.17.0
copulas==0.12.2
tqdm==4.67.1
seaborn==0.13.2

cd projet_itc
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

Remarque : sdv installe TVAE/CTGAN via sdv.single_table. torch doit être compatible avec la machine (CPU/GPU).

1) Clustering des instances réelles — auto_cluster_csv.py

Objectif : Explorer les clusters d'ITC-2007 (notamment la séparation Erlangen vs autres), produire des affectations et des scores, et un UMAP coloré.

Entrées attendues :

- data/real/real.csv (17 colonnes : name + 16 numériques).

Sorties :

- outputs/results/best_cluster_assignment.csv (étiquettes de cluster par instance)
- outputs/results/all_cluster_scores.csv (table des scores pour chaque algo de clustering)
- outputs/figures/best_clusters_umap.png (UMAP coloré par cluster)

Commandes :

```
cd projet_itc
python scripts/auto_cluster_csv.py \
  --real data/real/real.csv \
  --outdir outputs/results \
  --top 5 \
  --workers 8
```

Notes :

- Le script sauvegarde aussi un X_clean.csv et un X_pca.npy utiles pour debug.
- L'UMAP est généré avec les hyperparams par défaut du script.

2) Génération ITC-2007 — CTGAN et TVAE (CSV→CSV)

2.0) (Optionnel) Scinder Erlangen / non-Erlangen

Objectif : Crée deux fichiers si tu veux des entraînements séparés

Commande : python scripts/split_erlangen.py

2.1) Entraîner & générer — CTGAN

Commande : python scripts/train_ctgan.py

Ce que fait le script :

- Charge data/real/real.csv (ou la variante “no_erlangen”).
- Détecte le schéma via SingleTableMetadata.
- Entraîne CTGANSynthesizer (SDV).
- Échantillonne par lots jusqu’à N_SYN valides, en appliquant le filtrage de validité de De Coster (cohérences, bornes, etc.).
- **Sauvegarde :**
  - models/ctgan_model.pkl
  - outputs/ctgan/metadata.json
  - data/synthetic/synthetic_ctgan.csv

2.2) Entraîner & générer — TVAE

Commande : python scripts/train_tvae.py

Ce que fait le script :

- Même pipeline que CTGAN mais avec TVAESynthesizer.
- Sauvegarde :
  - models/tvae_model.pkl
  - outputs/tvae/metadata.json
  - data/synthetic/synthetic_tvae.csv

3) Conversion CSV → ECTT (pour tests solveur ITC-2007)

Objectif : Convertir chaque ligne d'un CSV synthétique (CTGAN ou TVAE) en .ectt pour pouvoir lancer un solveur ITC-2007.

Entrées :

- data/synthetic/synthetic_ctgan.csv (ou synthetic_tvae.csv)

Sorties :

- data/synthetic/ectt/*.ectt

Commande :

```
python scripts/csv2ectt.py \
 --csv data/synthetic/synthetic_ctgan.csv \
 --outdir data/synthetic/ectt
```

Principe :

- Parcourt les lignes du CSV synthétique.
- Typage/arrondis cohérents.
- Écrit un .ectt par instance (nommés syn_XXXXX.ectt).

4) Lancer le solveur ITC-2007 et visualiser (résolu / non résolu)

Objectif. Évaluer, sur un lot d'instances .ectt, le pourcentage d'instances résolues < 300 s par le solveur ITC-2007 de Müller, puis projeter en UMAP les instances (réelles + synthétiques) en colorant selon le statut (résolue / non résolue)

Entrées :

- data/synthetic/ectt/ : instances ECTT générées (via csv2ectt.py).
- data/real/real.csv : instances réelles.
- data/synthetic/synthetic_ctgan.csv : instances synthétiques.

Sorties :

- outputs/results/solve_ctgan.csv : récapitulatif solveur (instance, statut, temps, etc.).
- outputs/figures/umap_ctgan_solved.png : UMAP réel + synthétique (résolu/non).

4.1) Lancer le solveur

```
python scripts/run_solver_and_plot.py \
  --ectt_dir data/synthetic/ectt \
  --feat data/synthetic/synthetic_ctgan.csv \
  --output_dir outputs/results/solver_ctgan \
  --out_csv outputs/results/solve_ctgan.csv \
  --seed 42 \
  --timeout 300 \
  --jobs 8
```

4.2) Visualiser en UMAP (résolu / non résolu)

```
python scripts/show_umap_solved.py \
  --real data/real/real.csv \
  --syn  data/synthetic/synthetic_ctgan.csv \
  --sol_csv outputs/results/solve_ctgan.csv \
  --out  outputs/figures/umap_ctgan_solved.png
```

Principe :

- Le script run_solver_and_plot.py parcourt les .ectt, lance le solveur avec timeout (par défaut 300 s), journalise temps et statut.
- Le script show_umap_solved.py charge réelles + synthétiques, calcule la projection UMAP et superpose le statut “résolue / non résolue” à partir du CSV solveur.

5) Génération ITC-2019 (XML) par algorithme génétique

8.1 Organisation & prérequis

Arborescence recommandée (rappel, complétée pour 2019)

```
projet_itc/
├── data/
│   ├── real/
│   │   └── real.csv
│   ├── synthetic/
│   │   └── ectt/
│   └── itc2019/
│       └── *.xml              # Instances réelles ITC-2019
├── models/
├── outputs/
│   ├── results/
│   └── figures/
├── scripts/
│   ├── ga_itc2019.py          # (GA ITC-2019)
│   ├── analyse_ga.py          # (analyse des résultats GA)
│   └── compare_instances.py   # (comparaison réal vs synth XML)
├── solver/
│   └── itc2019/
│       ├── itc2019.jar        # JAR principal (cpsolver-itc2019)
│       ├── dependency/        # * si requis par le JAR
│       │   └── *.jar
│   └── configuration/
│   └── default.cfg
├── scripts/...(autres)
├── requirements.txt
└── README.md
```

8.2 Script principal — Algorithme génétique (ITC-2019) : scripts/ga_itc2019.py

Objectif :

Générer in-situ des instances XML synthétiques proches des réelles, en pilotant la difficulté via le temps d'initialisation cible mesuré par le solveur ITC-2019. À chaque génération, l'algorithme produit n enfants par mutations, puis sélectionne le meilleur (fitness = |temps_enfant – temps_cible|).

Commande :

python scripts/ga_itc2019.py

Sorties attendues :

- outputs/ga_instances/<instance>/...xml : meilleurs enfants retenus par génération
- outputs/ga_instances/ga_results.csv : log consolidé (colonnes base_instance, generation, mutations_applied, solve_time_s, target_time_s, fitness_score)

Notes & pièges :

- Erreur initiale corrigée : la fitness est bien |temps_enfant – temps_cible|, pas le temps direct affiché.
- Nettoyage disque : le code supprime les enfants non retenus, limitant l'usage disque.
- Temps cible : défini par instance réelle au départ (solve_time sur l'originale).

- **Stratégies :**
  - structure : mutations « logiques », préserve très fortement la structure.
  - agressive : hausse des proba d'opérateurs impactants + suppressions plus fréquentes.
  - physique : focus salles & distances (stabilité très élevée sur cas faciles).

8.3 Analyse des résultats GA : scripts/analyse_ga.py

Objectif :

Lire ga_results.csv, catégoriser les instances réelles par temps cible (Très rapide / Rapide / Moyen / Long / Très long), mesurer l'évolution par génération et produire :

- un résumé textuel (analyse_summary.txt),
- un graphique d'évolution evolution_comparative_par_categorie.png.

Commande :

python scripts/analyse.py

Sorties attendues :

- outputs/figures/evolution_comparative_par_categorie.png
- outputs/results/analyse_summary.txt

8.4 Comparaison structurelle réal vs synth (moyenne) : scripts/compare_instances.py

Objectif :

- Comparer en lot chaque instance réelle XML à la moyenne de ses enfants générés (dans son sous-dossier), via des statistiques structurelles niveau 1 (comptages) et numériques niveau 2 (capacités moyennes, pénalités moyennes, etc.).
- Produire un CSV détaillé et un résumé simple avec score de différence (%).

Commande :

python scripts/compare_instances.py

Sortie attendue :

- outputs/results/comparison_summary_average.csv (avec, pour chaque instance, les comptes et moyennes réelles vs synthétiques, et un score de différence agrégé).