Bit Packing Compression
Software Engineering Project 2025

Rapport de projet présenté par :
Mohamed El Amine MAZOUZ
Master 1 - Intelligence Artificielle

Professeur : Pr. Régin

Date : 2 novembre 2025

UNIVERSITÉ CÔTE D'AZUR

Table des matières

UNIVERSITÉ CÔTE D'AZUR

1. Introduction

Le présent rapport décrit le développement et l'évaluation d'un système de compression d'entiers basé sur le principe du Bit Packing. Ce projet a été réalisé dans le cadre du module de Software Engineering Project 2025, encadré par le Pr. Régin. L'objectif principal est de concevoir une méthode efficace pour réduire la taille des tableaux d'entiers lors de leur transmission, tout en préservant un accès direct à chaque élément compressé. Cette approche est particulièrement pertinente dans les contextes où la bande passante réseau est limitée et où les opérations de lecture doivent rester rapides.

2. Présentation du problème

La transmission d'ensembles d'entiers est une tâche fréquente dans de nombreux systèmes informatiques et applications réseau. Cependant, représenter chaque entier sur 32 bits est souvent excessif lorsque les valeurs manipulées sont petites. L'idée du Bit Packing consiste à n'utiliser que le nombre minimal de bits nécessaires pour représenter les données, afin de réduire la taille totale transmise et ainsi améliorer la vitesse de communication.

3. Conception et architecture du projet

Le projet repose sur une architecture modulaire comportant plusieurs classes de compression implémentant une interface commune. Une fabrique (Factory) permet de sélectionner dynamiquement la méthode de compression souhaitée parmi plusieurs variantes :

- CrossBoundaryPacker : autorise les valeurs à traverser les frontières de mots 32 bits.
- NoCrossPacker : interdit ce chevauchement pour simplifier la décompression.
- OverflowBitPacker : gère une zone d'overflow pour isoler les valeurs exceptionnelles.

4. Implémentation des méthodes de compression

Les classes de compression ont été développées en Python sans dépendances externes. Chaque implémentation assure trois fonctions principales :

- compress(array) : compresse un tableau d'entiers.
- decompress(array) : décompresse dans un tableau préalloué.
- get(i) : renvoie le i-ème entier original sans décompression complète.

Deux stratégies sont comparées :

1. Écriture avec chevauchement (cross), permettant une meilleure compacité.
2. Écriture sans chevauchement (nocross), privilégiant la simplicité d'accès.

5. Mesures de performance et analyse

Des mesures précises ont été effectuées pour évaluer le temps de compression, de décompression et d'accès direct (get). Les expériences ont été réalisées sur un ensemble de 100 000 entiers, avec une valeur maximale de 4095 (12 bits nécessaires). Les résultats sont les suivants :

Méthode	k (bits)	Ratio	Temps comp. (ms)	Temps décomp. (ms)
CrossBoundary	12	0.375	40.45	49.01
NoCross	12	0.375	36.38	46.81
Overflow-Cross	13	0.406	114.39	84.52

(Résultats de mon ordi personnel, ça peut varier selon plusieurs conditions)

L'analyse montre que les deux premières méthodes (cross et nocross) offrent un excellent ratio de compression (37,5 %) avec des temps de traitement inférieurs à 50 ms. En revanche, la méthode Overflow-Cross, bien que plus flexible, présente un surcoût temporel notable dû à la gestion de la zone d'overflow.

Le débit de seuil (break-even bitrate) calculé montre que la compression devient bénéfique dès que la bande passante est inférieure à environ 22 Mbits/s pour le mode cross et 24 Mbits/s pour le mode nocross.

6. Problèmes rencontrés et solutions

Plusieurs difficultés ont été rencontrées durant la mise en œuvre :

- Gestion du chevauchement des bits entre deux entiers consécutifs.
- Conversion correcte entre entiers signés et non signés.
- Mesure précise du temps d'exécution (optimisation du protocole expérimental).
- Ces problèmes ont été résolus en adoptant une gestion binaire rigoureuse et en utilisant la fonction time.perf_counter_ns() pour des mesures fiables à la nanoseconde près.

7. Tests et validation

L'ensemble des fonctionnalités a été validé à l'aide de tests unitaires (pytest). Les cinq tests couvrant les modes cross, nocross, overflow et les deux variantes de codage signé (ZigZag et complément à deux) ont tous été validés. Les temps d'exécution des tests sont inférieurs à 0,05 seconde, ce qui confirme la robustesse de l'implémentation.

8. Analyse critique

Le projet remplit pleinement les objectifs fixés. La méthode de compression est efficace et conserve un accès direct aux éléments. Toutefois, certaines optimisations restent envisageables, notamment dans la gestion dynamique de la taille des blocs et la vectorisation des opérations pour accélérer la compression. L'approche Overflow-Cross, bien qu'un peu plus lente, ouvre la voie à des stratégies hybrides plus intelligentes.

9. Conclusion et perspectives

Ce projet a permis de concevoir et d'évaluer un système complet de compression Bit Packing pour entiers. Les résultats expérimentaux démontrent l'efficacité de la solution, particulièrement dans les environnements à faible bande passante. À l'avenir, des extensions pourraient inclure la prise en charge de flux de données en continu (streaming) et l'intégration d'algorithmes adaptatifs capables d'ajuster automatiquement le nombre de bits utilisés selon la distribution des données.

10. Références

- Documentation officielle Python : https://docs.python.org/3/
- Spécifications du projet : Software Engineering Project 2025, Pr. Régin
- ChatGPT