Activités, problèmes, résultats

Clément Legrand-Lixon

22 Mai 2018 Arrivée dans le laboratoire (Cristal). Rencontre de l'équipe (doctorants et stagiaires). Installation du poste de travail (Ubuntu, espace de travail : VSC, latex, R). Découverte du problème, recherche d'informations (variantes : multi-dépôts, avec capacité, avec limite de temps...). Lecture des articles *A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem* (K. Altınel, T. Oncan), et *A simple, deterministic, and efficient knowledge-driven heuristic for the vehicle routing problem* (Florian Arnold, Kenneth Sorensen). Vu administration pour la gratification de stage.

23 Mai 2018 Fin de lecture des articles, résumé des articles tapés en tex. Implémentation de l'algorithme de Clarke and Wright (C W), en python. Mise en place d'une heuristique basique (avec paramètre λ). Tests de l'algo sur des instances fabriquées aléatoirement. Tentative pour trouver une valeur optimale pour le paramètre selon la taille de l'entrée. Résultats disponibles avec la figure 1.

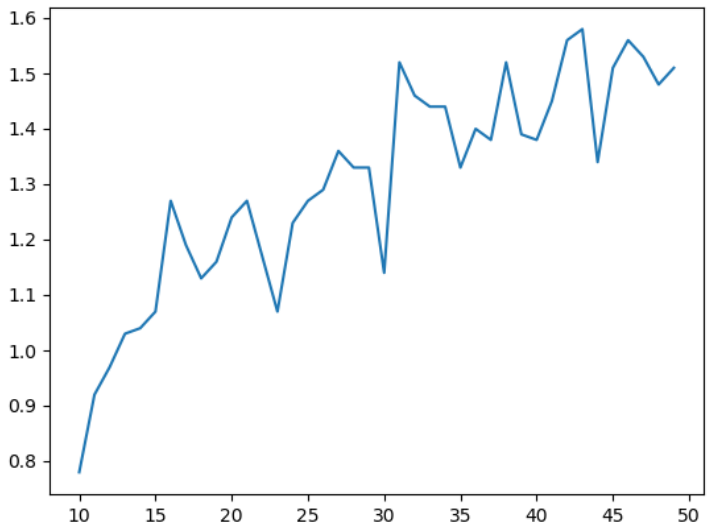

FIGURE 1 – Valeur de lambda optimale

24 Mai 2018 Rencontre avec encadrante senior : Marie-Éléonore, bilan sur les premiers jours. Obtention du code C++ de l'heuristique de CW (complète, mais quelques erreurs présentes à corriger). Objectif : Pour la semaine prochaine, faire une présentation reprenant la nouvelle

heuristique développée dans l'article de F. Arnold et K. Sorensen. Et éventuellement implémentation des opérateurs locaux utilisés dans l'heuristique en expliquant les intérêts et limites de chaque (complexité, avantage...). Début de la présentation. Recherche d'articles détaillant les opérateurs utilisés. Peu de résultats (page wikipédia pour Lin Kernighan).

25 Mai 2018 Rencontre avec Lætitia Jourdan (responsable de l'équipe). Implémentation de l'heuristique de Lin-Kernighan : pour une tournée donnée, optimise la visite des clients (utilisée pour TSP). Commence par réaliser 2-opt (cherche un changement entre deux arêtes qui améliore la tournée). Si trouvé on passe 3-opt (echange de 3 aretes), jusqu'à k-opt (k choisi). Puis applique la meilleure modif (la meilleure i-opt). S'arête lorsque plus d'améliorations possibles).

Implémentation de 2-opt en python, tests sur quelques instances. On part de 2, et en appliquant 2-opt on obtient 3. 'Dépôt représenté en bleu).

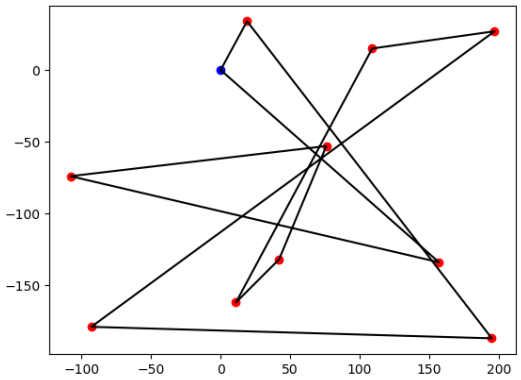

FIGURE 2 – Instance initiale

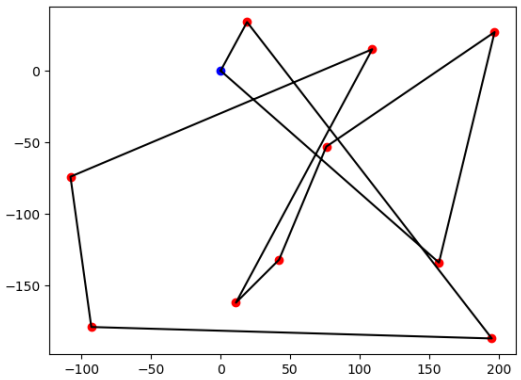

FIGURE 3 – Optimisation avec 2-opt

Rencontre de problèmes avec des instances supérieures, et LK. → A résoudre lundi. Opérateur à utiliser principalement sur des petites portions de route, complexité élevée $O(n^k)$ avec n le nombre de clients.

28 Mai 2018 Objectifs : finir d'implémenter LK, tester, et finir le développement dans la présentation. Améliorer présentation. Implémenter un autre opérateur *Cross-exchange* ou *Ejection-chain*.

Réalisation : LK corrigé (fonctionne avec 2-opt... généraliser à k-opt ?). Tests réalisés sur la tournée en image 4, où on obtient la tournée en image 5.

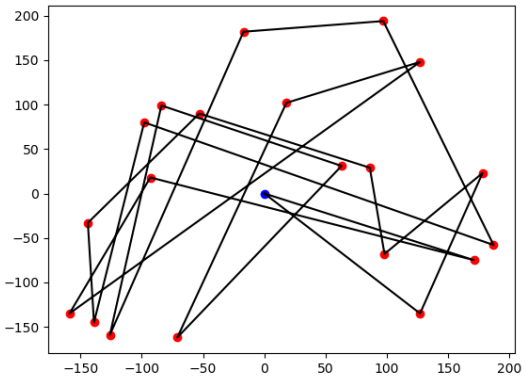

FIGURE 4 – Instance initiale pour 20 clients

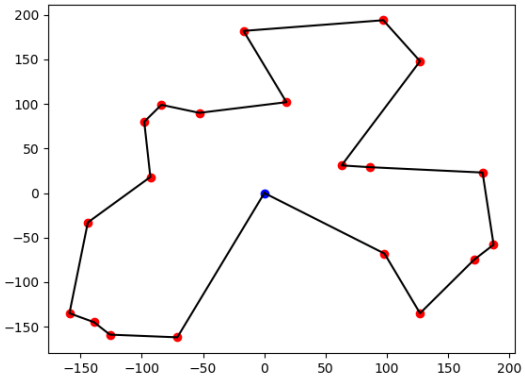

FIGURE 5 – Solution obtenue avec Lin-Kernighan

Création (enfin !) d'un dépôt GIT, pour partager mes documents (travail, présentations...)

Début implémentation de Cross-exchange (local). Article détaillant cet opérateur : *A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows*. Cet opérateur consiste en l'échange de toute séquence de clients entre 2 tournées. L'aspect local est d'autant plus intéressant, que la complexité passe en quadratique dans le pire cas, au lieu de $O(n^4)$. On commence par calculer k pp voisins pour chaque nœud (pré-calcul). Après avoir choisi une arête (a, b), on regarde les plus proches voisins du premier nœud, on prend le premier qui est sur une tournée différente, (c, v_a). On construit les deux arêtes (a, v_a) et (c, b). Puis on essaye d'échanger $(v_a + 1, v_a + 2)$, avec $(b + 1, b + 2)$, ou $(b + 2, b + 3)$ ainsi de suite jusqu'à retomber sur l'arête de départ ou trouver une amélioration. En pratique on trouve rapidement une amélioration d'où une complexité en général linéaire. Il faut aussi veiller à ne pas dépasser les contraintes de capacité (non prises en compte dans un premier temps). Fin implémentation,

tests réalisés sur une instance particulière (cf ...), 3 exécutions ont été réalisées sur des arêtes différentes, (arête choisie en rouge). Exemple avec l'image 6, on obtient le résultat sur l'image 7.

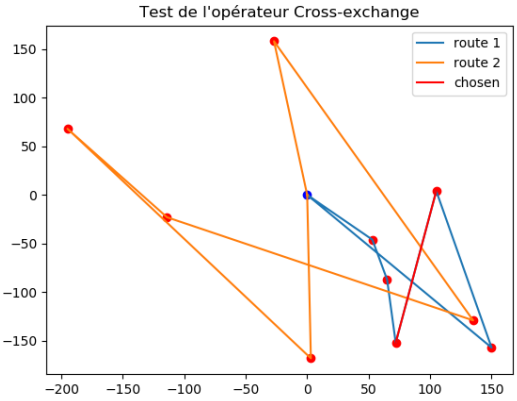

FIGURE 6 – Instance initiale pour 10 clients répartis sur 2 tournées

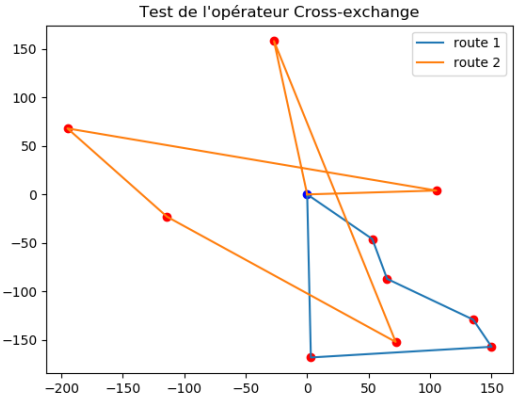

FIGURE 7 – Amélioration obtenue avec Cross-exchange

29 Mai 2018 Objectifs : Corriger présentation pour cross-exchange. Faire implémentation de ejection-chain, tester. Finir présentation ? Finir de corriger code C++ ?

Réalisation : Ajout de cross-exchange dans la présentation. Implémentation de l'ejection-chain, reste quelques problèmes (ajout/suppression de routes, manque les capacités). Ajout des capacités pour le cross-exchange, et découverte problème dans implémentation → correction et nouveaux tests. Epuration code et homogénéisation des notations. Commentaires ajoutés. Tests pour correction de CE (et ajout de capacités) :

Tests pour ejection-chain (sans capacités) : initialement avec image 10, pour obtenir l'image 11 (21 clients, 3 routes, 15 replacements).

30 Mai 2018 Fin implémentation ejection-chain (avec gestion de capacité), création et suppression de route à faire. Ajout à la présentation, et fin (?) de celle-ci. Re-

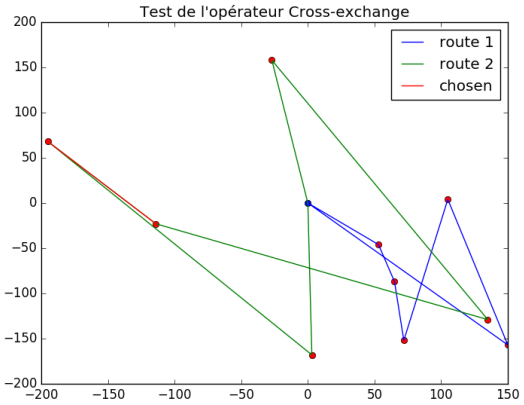

FIGURE 8 – Instance initiale pour 10 clients répartis sur 2 tournées

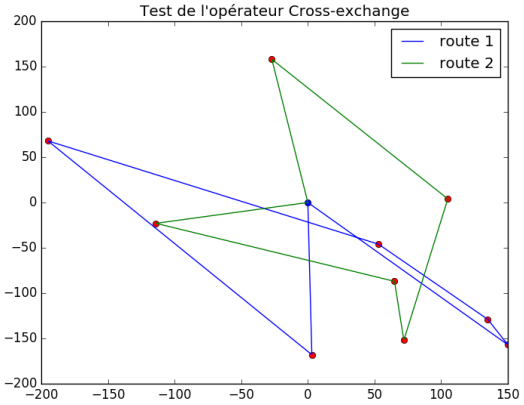

FIGURE 9 – Amélioration obtenue avec Cross-exchange

correction du cross-exchange (et cette fois c'est bon !, les clients échangés initialement n'étaient pas bons). Article pour ejection-chain dans vrp with time-window : *Fast Ejection Chain Algorithms for Vehicle Routing with Time Windows* de Herman Sontrop, Pieter van der Horn, and Marc Uetz. Regroupement des opérateurs, et début implémentation de l'heuristique (fonction de pénalisation, recuperation de la pire arête, execution des opérateurs). Premier test (beaucoup d'erreurs...). Correction, principal problème : opérateurs pas utilisables si beaucoup de tournées (CE, n'est censé géré que 2 tournées...).

31 Mai 2018 Correction heuristique, ajout de l'optimisation globale, reset et update des fonctions de pénalisation. Tests sur différentes instances avec capacité totale variable. Premiers tests plutôt concluants, calcul assez rapide (30 sec pour 100 clients). Résultats présents sur les images 12 et 13.

Remarques : toujours quelques erreurs au niveau du dépôt à cause de LK (par ailleurs qui se contente de 2-opt, à voir s'il est utile d'aller plus loin). Actuellement, l'heuristique ne choisit que des solutions améliorantes, il est possible de passer à côtés de grosses optimisations (ou alors capacités trop restrictives...).

Conférence, Blockchains et Cryptomonnaies, par Jean-Paul Delahaye : Présentation des

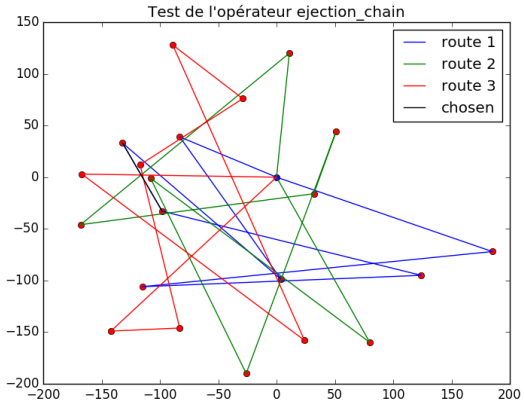

FIGURE 10 – Instance initiale pour 21 clients répartis sur 3 tournées

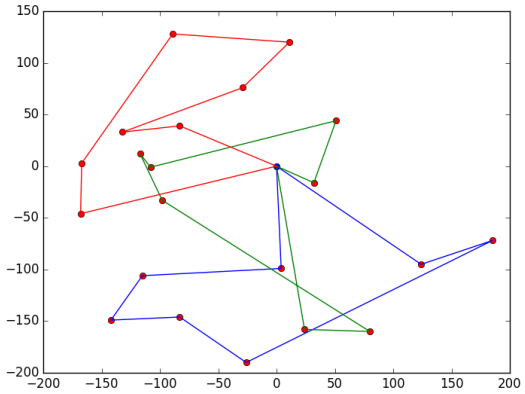

FIGURE 11 – Amélioration obtenue avec Ejection-chain (15 replacements)

notions (utilisation des Blockchains, développement sur les crypto-monnaies). Limites et future. Principales notes :

- Blockchain : fichier multiplié sur un réseau P2P (donc indestructible). On peut écrire dessus en respectant des contraintes définies sur le réseau.
- Cryptomonnaies (ou crypto-actif → pas vraiment une monnaie à cause de sa volatilité).
- Pour ajouter une page au blockchain du bitcoin (justification de transaction), il faut inverser partiellement une fonction de hash, pour que la page ajoutée ait la bonne signature. Contrainte sur la taille de la page 1 MO (suscite un fork en 2017 → division bitcoin et bitcoin cash). Preuve d'intérêt : toutes les 10 min un certain nb de bitcoins sont attribués (jusque 2140) à ceux qui ont trouvé la solution. Difficulté ajustée si + ou - de 10 min en moyenne, de sorte à ce que la puissance de calcul nécessaire soit toujours plus importante.
- Aujourd'hui minage = 3 centrales nucléaires (dont la puissance principale est en Chine). Cette quantité démentielle d'electricité pose problème. (il faut effectuer de l'ordre de 10^{18} calculs pour miner le bitcoin en 10 min.
- Ces monnaies (même toutes réunies) ne peuvent concurrencer l'or sur le marché financier (8000 milliards de dollars pour l'or contre 300 milliards pour les crypto-actifs).
- D'autres crypto-monnaies voient le jour, moins énergivores : preuve d'enjeu.

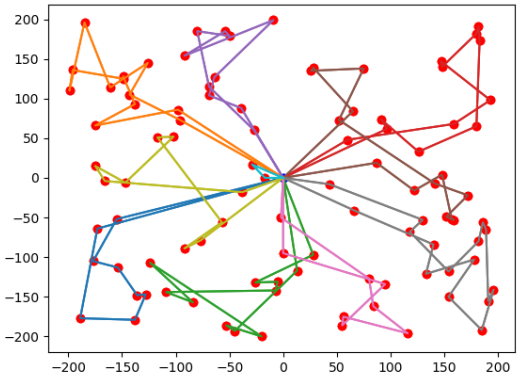

FIGURE 12 – Instance initiale pour 100 clients, avec Clark and Wright

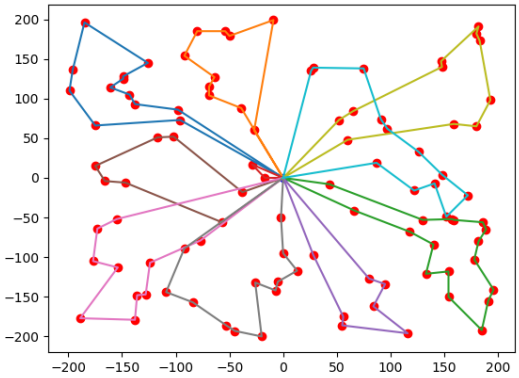

FIGURE 13 – Amélioration obtenue avec l'heuristique

— Vulnérabilités possibles (prédiction Adi Shamir) : attaques sur les courbes elliptiques, ou fonctions de hash ; attaque à 51%, qqun possède plus de la moitié de la puissance de calcul du réseau.
— Pas assez de transactions par seconde (environ 4/s), pour etre adapté au marché financier (comme Paypal).

Bilan réunion : Ajout/supp de routes inutiles à priori pour ce qu'on veut faire. Améliorer l'algo CW (au moins un 2 opt supp). Retravailler la partie sur la pire arête dans l'article. Utiliser les instances de la littérature pour les tests. Objectif principal semaine prochaine : trouver (?) un lien entre solutions améliorées et solutions CW.

1 Juin 2018 Amélioration de l'heuristique en retravaillant EC, maintenant parcours de l'espace de solutions autour d'une solution globale tant qu'on trouve pas mieux (EC plus flexible). Evaluation de l'amélioration. Amélioration de la solution initiale, et comparaison entre celle-ci et solution obtenue avec l'heuristique. Besoin de refactoriser le code et de l'alléger, modifications pas toujours claires...