


Résumé : Optimisation distribuée pour la recharge de véhicules électriques

 Xinyu HUANG

Master en Androïde, Sorbonne université

1 MOTS CLÉS

Frank&Wolfe algorithm, problème de recharge intelligente, optimisation convexe

2 INTRODUCTION

5 Ce stage, concernant un projet de recherche financé par le Programme Gaspard Monge pour l'Optimisation (PGMO), est supervisé conjointement d'un côté par Laurent Pfeiffer, chercheur au L2S, INRIA-Saclay, et de l'autre côté par Nadia Oudjane, Cheng Wan et Guilhem Dupuis, chercheurs au EDF Lab Saclay.
10 Le stage consist à adapter l'algorithme Stochastic Frank&Wolfe à un problème de rechargement des véhicules électriques et aussi l'amélioration sur le problème et l'algorithme.

3 ETAT DE L'ART

15 Pendant ce stage, j'étudie un problème d'optimization agrégative, défini comme:

$$\inf_{x \in \mathcal{X}} J(x) = f\left(\frac{1}{N} \sum_{i=1}^N g_i(x_i)\right) + \frac{1}{N} \sum_{i=1}^N h_i(x_i) \quad (\mathcal{P})$$

where $x = (x_1, \dots, x_N) \in \mathcal{X} = \prod_{i=1}^N \mathcal{X}_i$ and where the following data is given:

- $g_i : \mathcal{X}_i \rightarrow \mathbb{R}^n, i = 1, \dots, N$
- $h_i : \mathcal{X}_i \rightarrow \mathbb{R}, i = 1, \dots, N$
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

The problem can be interpreted as a multi-agent model in which

- N is the number of agents
- \mathcal{X}_i is the decision set of agent i
- $h_i(x_i)$ is individual cost function of agent i
- $g_i(x_i)$ is contribution of agent i to a common good
- $\frac{1}{N} \sum_{i=1}^N g_i(x_i)$ describes a common good, referred to as aggregate
- f : a social cost associated with the aggregate.

20 Les développements algorithmiques proposés dans ce rapport reposent principalement sur l'algorithme Stochastic Frank-Wolfe (SFW) présenté dans l'article [Liu et al. 2022](#) qui est l'adaptation de l'algorithme Frank&Wolfe ([Frank and Wolfe 1956](#)) à ce problème. La particularité de l'algorithme SFW proposé dans [Liu et al. 2022](#) est sa capacité à traiter la non-convexité de la fonction de coût J (seule la fonction f doit être convexe). Il s'agit d'un algorithme itératif qui nécessite uniquement la résolution de sous-problèmes, spécifiques à chaque agent, de complexité modérée.

4 DÉVELOPPEMENT DES ALGORITHMES

40 L'algorithme SFW peut donner un résultat approximative mais il exist toujours une différence non négligeable entre les valeurs optimales produites par le solveur (CPLEX) et celles générées par l'algorithme SFW. Donc les variants algorithmiques sont proposés:

4.1 Classical Frank&Wolfe

Nous proposons d'utiliser l'algorithme SFW sur un nombre spécifié d'itérations, puis d'exploiter la convexité partielle du problème : l'algorithme classique de Frank&Wolfe s'applique au problème obtenu en figeant les variables binaires. δ_i^k est le coefficient pour la combinaison convexe de l'algorithme Frank&Wolfe. Il exist trois variants selon le différent choix du stepsize δ_i^k .

- CFW1 : $\delta_{i,k} = 2/(k+2)$, le choix standard dans la littérature.
- CFW2 : Nous proposons de prendre un stepsize qui soit optimale et uniforme par rapport à i , c'est-à-dire, nous définissons $\delta_i^k = \delta^k$, où δ^k est la solution de :

$$\inf_{\delta \in [0,1]} J((1-\delta)\bar{x}^k + \delta x^k).$$

- CFW3 : Finalement on propose un stepsize spécifique pour chaque agent, qui est possible car il y a une ensemble faisable pour chaque agent. Donc on défini $(\delta_i^k)_{i=1,\dots,N}$ comme la solution de:

$$\inf_{\delta \in [0,1]^N} J\left((1-\delta_1)\bar{x}_1^k + \delta x_1^k, \dots, (1-\delta_N)\bar{x}_N^k + \delta x_N^k\right)$$

4.2 Méthode gloutonne(SFW2)

Nous proposons une approche gloutonne consistant à vérifier si la solution s'améliore en modifiant les valeurs d'un agent particulier. Étant donné que nous parcourons tous les agents avec une probabilité uniforme $\delta_k = \frac{2}{k+2}$, nous essayons d'injecter les nouvelles valeurs d'un agent dans notre solution et ne conservons cette solution que si elle est meilleure.

4.3 Méthode robuste

La méthode robuste a été proposée après les premiers tests. Nous avons constaté que la méthode gloutonne est très rapide au démarrage et que la méthode CFW3 est généralement très performante, capable d'atteindre une bonne valeur après 100 itérations. Notre méthode robuste consiste simplement à combiner les avantages de ces deux méthodes : nous commençons par SFW2, et si, après cinq itérations, aucune amélioration n'est observée, nous basculons vers la méthode CFW3. Cette méthode est appelée "robuste" car elle surpasse toutes les autres méthodes en termes de valeur optimale et de temps d'exécution.

5 FORMULATION DES DIFFÉRENTS PROBLÈMES

Après avoir exploré plusieurs variantes algorithmiques de l'algorithme SFW, nous avons identifié une autre voie pour une amélioration potentielle. Cela consiste à optimiser les variables continues associées au "Service", ce qui peut être fait de manière analytique. Nous faisons référence à l'optimisation des variables de service continues comme le "booster", que nous proposons d'appliquer après chaque mise à jour de la solution. De plus, nous avons également incorporé ce "booster" dans le problème original, le transformant ainsi en un nouveau problème présentant une structure agrégative, que nous appelons le "problème réduit".

6 CONCLUSION

Pendant le stage, une séquence d'améliorations algorithmiques a été étudiée. Nous avons réussi à implémenter un algorithme robuste pour notre problème. Il reste encore beaucoup d'améliorations à faire suite à ce que j'ai implémenté. Par exemple, il est possible de tester des instances encore plus complexes, de résoudre les sous-problèmes en parallèle, ou bien de les tester dans un autre langage de programmation plus adapté à la programmation mathématique, comme Julia.

Ici, nous présentons les pseudo-codes des algorithmes de SFW, FW et de la méthode gloutonne. La méthode robuste reste triviale car elle est la combinaison des deux méthodes présentées ici.

Algorithme 1 : Stochastic Frank-Wolfe Algorithm

Input : $\bar{x}^{(0)}$: vector containing $(\bar{x}_1^{(0)}, \dots, \bar{x}_i^{(0)}, \dots, \bar{x}_N^{(0)})$, each representing the state of an agent
 $(n_k)_{k \in N}$: a list of integers, where n_k represents the number of drawings for the k -th iteration.

```

1 for  $k \leftarrow 1$  to  $K$  do
2   Set  $\lambda = \nabla f(\frac{1}{N} \sum_{i=1}^N g_i(\bar{x}_i))$ ;
3   for  $i \leftarrow 1$  to  $N$  do
4     Find a solution  $x_i^k$  to
        
$$\inf_{x_i^k \in X_i} \langle \lambda, g_i(x_i) \rangle + h_i(x_i)$$

5    $x^k = (x_1^k, \dots, x_i^k, \dots, x_N^k)$ ;
6   Set  $\delta_k = 2/(k+2)$ ;
7   for  $j \leftarrow 1$  to  $n_k$  do
8     for  $i \leftarrow 1$  to  $N$  do
9       Draw the state of each agent  $\tilde{x}_i^j$  according to
          
$$\tilde{x}_i^j = \begin{cases} \bar{x}_i^k & \text{if } 0 < Z_{i,j,k} \leq 1 - \delta_k \\ x_i^k & \text{if } 1 - \delta_k < Z_{i,j,k} \leq 1, \end{cases}$$

          Here  $Z_{i,j,k}$  is a random variable with uniform distribution in  $[0, 1]$ .
10      Set  $\tilde{x}^j = (\tilde{x}_1^j, \dots, \tilde{x}_i^j, \dots, \tilde{x}_N^j)$ ;
11      Find  $\bar{x}^{(k+1)} \in \bigcup_{j \in \{1, \dots, n_k\}, x \in \{\tilde{x}_1^j, \dots, \tilde{x}_N^j\}} J(x)$ ;

```

Output : \bar{x}^{K+1} , the approximate solution

Algorithme 2 : Classical Frank&Wolfe

Input : $\bar{x}^{(K_0)}$: vector containing $(\bar{x}_1^{(k)}, \dots, \bar{x}_i^{(k)}, \dots, \bar{x}_N^{(k)})$, each representing the state of an agent, solution after K_0 iterations of SFW.

```

1 for  $k \leftarrow K_0$  to  $K$  do
2   Decompose  $\bar{x}^k$  as  $(\bar{x}_{int}^k, \bar{x}_{con}^k)$ , corresponding to binary and continuous variables.
3   Set  $\lambda = \nabla f(\frac{1}{N} \sum_{i=1}^N g_i(\bar{x}_{i,int}, \bar{x}_{i,con}))$ ;
4   for  $i \leftarrow 1$  to  $N$  do
5     Find a solution  $x_i$  to
        
$$\inf_{\substack{x_i = (x_{i,int}, x_{i,con}) \in X_i \\ x_{i,int} = \bar{x}_{i,int}^k}} \langle \lambda, g_i(x_i) \rangle + h_i(x_i)$$

6   Set  $x^k = (x_1, \dots, x_i, \dots, x_N)$ ;
7   Find  $\delta_i^k \in [0, 1]$ ,  $i = 1, \dots, N$ ;
8   for  $i \leftarrow 1$  to  $N$  do
9      $\bar{x}_i^{k+1} = (1 - \delta_i^k)\bar{x}_i^k + \delta_i^k x_i^k$ ;
10  Set  $\bar{x}^{k+1} = (\bar{x}_1^{k+1}, \dots, \bar{x}_i^{k+1}, \dots, \bar{x}_N^{k+1})$ ;

```

Output : \bar{x}^{K+1} , the approximate solution

Algorithme 3 : Greedy method

Input : $\bar{x}^k \in X$: the current approximate solution,
 $x^k \in X$: the solution to the sub-problems.

```

Set  $\delta_k = \frac{2}{2+k}$ .
1 Set  $\bar{x}_{new}^k = \bar{x}^k$ .
2 Set  $S_{new} = J(\bar{x}^k)$ .
3 for  $i \leftarrow 1$  to  $N$  do
4   Generate a random variable  $Z_{i,k}$  with uniform distribution over  $[0, 1]$ .
5   if  $Z_{i,k} < 1 - \delta_k$  then
6     Set  $\bar{x}_{new,i}^k = x_i^k$ ;
7     if  $J(\bar{x}_{new}^k) < S_{new}$  then
8       Set  $S_{new} = J(\bar{x}_{new}^k)$ .
9     else
10      Set  $\bar{x}_{new}^k = \bar{x}_i^k$ .
11 Set  $\bar{x}^{k+1} = \bar{x}_{new}^k$ .
Output :  $\bar{x}^{K+1}$ , the approximate solution

```

B DISPONIBILITÉ DU CODE

optimisation recharge vehicule

REFERENCES

- Frank, M. and P. Wolfe (1956). “An algorithm for quadratic programming”. *Naval Research Logistics Quarterly* 3(1-2), pages 95–110. DOI: <https://doi.org/10.1002/nav.3800030109>.
- Liu, K., N. Oudjane, and L. Pfeiffer (2022). *Decentralized resolution of finite-state, non-convex, and aggregative optimal control problems*.