Mintc : affectation de transports en commun par l'algorithme Mint

Documentation de référence. Version parallèle, avec poids par type de lien.

1. Objet et périmètre

Mintc calcule l'affectation d'une matrice origine-destination sur un réseau de transport en commun décrit par des fréquences (intervalles entre véhicules). Il applique l'algorithme **Mint** (P. Palmier, CETE Nord Picardie). Mint reprend le principe des stratégies optimales de Spiess et Florian (1989) et corrige trois limites :

- **L1** : la répartition entre lignes attractives dépend aussi des temps de parcours, pas seulement des fréquences.
- **L2** : les liens de marche entrent dans la répartition comme des stratégies à fréquence infinie.
- **L3** : il est possible de rester dans un véhicule ou de descendre pour changer, selon ce qui coûte le moins.

Le programme produit :

- le volume, les embarquements et les descentes de chaque lien ;
- le temps attendu, le coût généralisé attendu et le nombre moyen d'embarquements de chaque OD ;
- les stratégies attractives de chaque lien, pour chaque destination (optionnel).

Le calcul est fait par destination. Les destinations sont traitées en parallèle sur plusieurs threads.

Ce document décrit le programme tel qu'il est livré. Il ne remplace pas l'article de référence : *Mint: proposition of a new transit assignment algorithm for frequency based networks* (Palmier).

2. Démarrage rapide

Ligne de commande

```
Mintc reseau.csv matrice.csv sortie parametres.txt [nb_threads]
```

Argument	Rôle
`reseau.csv`	Fichier réseau (section 3.1)
`matrice.csv`	Fichier matrice (section 3.2)
`sortie`	Préfixe des fichiers de résultats (section 7)
`parametres.txt`	Fichier de paramètres (section 3.3)
`nb_threads`	Optionnel. Nombre de threads. Par défaut : nombre de coeurs logiques. Plafonné au nombre de destinations.

Sans les quatre premiers arguments, le programme affiche le rappel d'usage et s'arrête.

Compilation

Application console C# en un seul fichier, `Program.cs`, espace de noms `Mintc`. Aucune dépendance externe. Elle a été compilée et testée avec le compilateur Mono `mcs`. Elle n'a pas été testée avec d'autres compilateurs.

Fichiers produits

`sortie_od.txt`, `sortie_aff.txt`, `sortie_times.txt`, `sortie_stg.txt`, `sortie_log.txt`.

3. Fichiers d'entrée

Tous les fichiers sont des fichiers texte, séparateur `;`, sans ligne d'en-tête. Les nombres sont lus avec la culture de la machine (voir section 9).

3.1 Réseau

Une ligne par segment de ligne ou lien de marche.

Col.	Nom	Type	Signification
1	`i`	texte	Nœud initial
2	`j`	texte	Nœud final
3	`line`	texte	Identifiant de ligne
4	`time`	réel	Temps de parcours, en minutes

Col.	Nom	Type	Signification
5	`hdwy`	réel	Intervalle entre véhicules, en minutes. **0 = lien de marche ou connecteur, sans attente**
6	`vcap`	réel	Capacité du véhicule. Lue, **non utilisée**
7	`allowboai`	entier	1 : embarquement autorisé au nœud initial. 0 : interdit
8	`allowalij`	entier	1 : descente autorisée au nœud final. 0 : interdite
9	`type`	texte	Optionnel. Type du lien, pour les poids par type (section 4)

Règles :

- Les nœuds sont créés automatiquement à partir des colonnes 1 et 2.
- Les espaces autour de `i`, `j`, `line` et `type` sont ignorés.
- Les lignes vides sont ignorées.
- Une ligne non vide avec moins de 8 champs arrête le programme, avec son numéro de ligne.
- Les restrictions des colonnes 7 et 8 sont appliquées.

Conventions de modélisation :

- `hdwy > 0` : segment de ligne de transport. Les segments qui se suivent avec le **même identifiant de ligne** correspondent à un trajet dans le même véhicule. Passer à un autre identifiant de ligne est un embarquement.
- `hdwy = 0` : marche, ou connecteur entre une zone et un arrêt.
- Une zone d'origine doit avoir **un seul lien sortant** si la demande doit se répartir entre plusieurs lignes à l'arrêt. Voir la section 9.

3.2 Matrice

Col.	Nom	Signification
1	`o`	Identifiant du nœud origine
2	`d`	Identifiant du nœud destination
3	`demande`	Volume de demande, réel

L'ordre des lignes est libre. Le calcul est regroupé par destination, dans l'ordre de première apparition de chaque destination. Les lignes vides sont ignorées. Une ligne non vide avec moins de 3 champs arrête le programme.

3.3 Paramètres

Les six premières lignes sont obligatoires, dans cet ordre, avec une valeur par ligne.

Ligne	Nom	Signification
1	`cwait`	Poids du temps d'attente. **Doit être > 0**
2	`cmap`	Poids du temps de marche (valeur globale)
3	`tboa`	Temps d'embarquement, en minutes
4	`cboa`	Poids de l'embarquement (valeur globale)
5	`output_strategies`	`true` ou `false` : écrire ou non `sortie_stg.txt`
6	`alg_parameter`	Paramètre de calcul : taille des buckets d'ordonnancement des liens

Lignes optionnelles, à la suite des six premières :

Format	Effet
`type;cmap;cboa`	Poids propres au type de lien `type` (section 4)
`tboa_dans_temps;1`	Ajoute `tboa` à la colonne `temps`. Par défaut : 0 (comportement de l'article)
`# commentaire`	Ligne ignorée. Les lignes vides le sont aussi

Une ligne optionnelle mal formée arrête le programme. Un type défini deux fois aussi.

Exemple de fichier :

```
1
2
2
1
true
10
# type;cmap;cboa
acces;6;1
marche;3;1
metro;2;0.5
bus;2;2
```

4. Coût généralisé et poids par type de lien

Le coût d'un trajet est la somme de quatre composantes :

Composante	Coût
Temps à bord (lien avec `hdwy > 0`)	`time` (poids 1)
Temps de marche (lien avec `hdwy = 0`)	`time × cmap`
Embarquement	`tboa × cboa`
Attente	`cwait × intervalle` (voir section 5)

Poids par type. Chaque lien peut porter un type (colonne 9). Une ligne `type;cmap;cboa` du fichier de paramètres définit des poids pour ce type :

- `cmap` s'applique au temps de marche **du lien parcouru**, selon le type de ce lien. Il permet de distinguer accès, sortie et marche de correspondance.
- `cboa` s'applique à la pénalité d'embarquement, selon le type **du lien embarqué**. Il permet de distinguer par exemple bus et métro.
- `tboa` reste global.

Un lien sans type, ou dont le type n'est pas listé, utilise le `cmap` et le `cboa` globaux (lignes 2 et 4). Un type est sensible à la casse. Un `cmap` donné à un type de transport, ou un `cboa` donné à un type de marche, n'a aucun effet.

Quand les poids changent-ils les stratégies ? L'attractivité et les parts de flux dépendent des *écarts* de coût entre alternatives. Une pénalité commune à toutes les alternatives d'un lien décale tous les coûts de la même quantité et ne change rien. `cboa` n'a donc d'effet que si les alternatives ont des nombres d'embarquements différents, et `cmap` que si elles marchent des durées différentes. Si `tboa = 0`, `cboa` n'a aucun effet.

5. Algorithme

5.1 Principe

Pour chaque destination, le réseau est parcouru à rebours à partir de la destination. Chaque lien reçoit :

- un **ensemble de stratégies attractives**, au plus une par ligne ;
- un **coût moyen attendu** jusqu'à la destination (`avg_cout`) et un **temps moyen attendu** (`avg_time`).

Les liens sont traités dans l'ordre d'un étiquetage par buckets : le prochain lien traité est pris dans le bucket non vide de plus petit indice. Un lien déjà traité peut être rouvert si une meilleure stratégie apparaît.

5.2 Une stratégie

Une stratégie d'un lien `u` décrit « prendre cette ligne ensuite ». Elle porte :

Champ	Sens
`line`	Ligne de la stratégie. Clé unique dans le lien : une ligne ne peut y figurer qu'une fois
`hdwy`	Intervalle de la ligne ; 0 pour une stratégie sans attente (marche)
`n`	Fréquence, `60 / hdwy` , en véhicules par heure
`c`	Coût minimal : coût moyen du pivot, plus le coût du lien, plus l'embarquement
`t`	Temps minimal correspondant, sans poids
`p`	Part de la demande qui suit cette stratégie
`pivot`	Lien suivant. -1 : fin du trajet à la destination

5.3 Formules

Notations : `n_k = 60 / hdwy_k` , `c_k` coût minimal de la stratégie `k` . Les sommes portent sur les stratégies de transport (`hdwy > 0`) du lien.

- **M** : `M = (60·cwait + Σ n_k·c_k) / Σ n_k` . C'est le coût maximal attendu quand les lignes sont combinées.

- **M0** : seuil d'attractivité du lien. C'est le minimum de `M`, du meilleur `c + hdwy·cwait` et du coût des stratégies de marche.
- **Attractivité** : une stratégie est attractive si `c ⩽ M0`. Les autres sont retirées. `M` et `M0` sont alors recalculés, jusqu'à stabilité.
- **Part d'une stratégie de transport** : `p_k = (M0 − c_k) / (hdwy_k · cwait)`. Elle dépend des temps, pas seulement des fréquences (limite L1).
- **Part d'une stratégie de marche** (`hdwy = 0`) : `p = (M − M0) · Σn / (60 · cwait)`. S'il n'y a pas de stratégie de transport, `p = 1`.
- **Coût moyen** : `avg_cout = Σ p_k · 0,5 · (c_k + M0)`.
- **Temps moyen** : `avg_time = Σ p_k · (t_k + 0,5 · (M0 − c_k) / cwait)`. Par défaut, ce temps ne comprend ni les pondérations ni `tboa`.

Les parts somment à 1.

Correspondance avec les notations de l'article. Dans le code, `M0` est le `M` de l'article (marche comprise) et `M` du code est le `M'` de l'article (hors marche). `c` est le temps minimal `μ`.

5.4 Création et optimisation des stratégies

- Un lien atteint pour la première fois reçoit une stratégie. S'il change de ligne, c'est un embarquement : la stratégie est construite à partir du coût moyen du pivot. Sinon, il hérite des stratégies du pivot, avec le temps et le coût de son propre lien en plus.
- Un lien déjà atteint peut être amélioré. Si la nouvelle stratégie d'une même ligne coûte moins que l'existante (`c` plus petit), elle la remplace. Si l'amélioration ne réduit pas le coût moyen du lien, le lien retrouve son état d'avant.
- **Indépendance des stratégies** : une ligne n'apparaît qu'une fois par lien.
- **Cycles** : un lien n'est ajouté que s'il n'appartient pas déjà aux liens qui composent les stratégies existantes.
- `allowboai` et `allowalij` limitent embarquements et descentes.

5.5 Chargement de la demande

Pour chaque OD :

1. La demande est placée sur le lien sortant de l'origine qui a le plus petit `avg_cout` parmi les liens atteints.
2. Elle est propagée de lien en lien selon les parts `p`, dans l'ordre décroissant de `pos`, vers le pivot de chaque stratégie.

3. Le volume est ajouté à chaque lien traversé. L'embarquement est compté quand le pivot est un lien de transport d'une autre ligne (ou le premier lien de transport). La descente est comptée quand on quitte une ligne de transport pour une autre, ou à la destination.

`ncorr` est le total des embarquements divisé par la demande : c'est le **nombre moyen d'embarquements** de l'OD. Un trajet direct sur une seule ligne vaut 1. Un trajet à pied vaut 0.

6. Exécution parallèle

- **Unité de travail** : une destination. Les calculs des destinations sont indépendants.
- **Copie du réseau par thread** : chaque thread travaille sur sa copie. Les nœuds, qui ne sont jamais modifiés, sont partagés.
- **Ordre des résultats** : les résultats sont écrits et cumulés dans l'ordre d'origine des destinations, quel que soit le nombre de threads. Les cinq fichiers de sortie sont identiques entre 1 et 8 threads (testé).
- **Mémoire** : une copie du réseau par thread, plus les résultats en attente d'écriture, limités à 4 fois le nombre de threads.
- **Erreur dans un thread** : le programme s'arrête avec un message qui reprend l'erreur d'origine.
- **Affichage** : `Destinations: k/N`, où `k` est le nombre de destinations terminées.
- **Précision** : les cumuls de `sortie_aff.txt` sont additionnés par destination. Ils peuvent différer de la version séquentielle d'environ 1e-6 en valeur relative, à cause de l'arrondi des `float`.

Le gain de vitesse dépend de la machine et du réseau. Il n'a pas été mesuré (l'environnement de test n'avait qu'un coeur).

7. Fichiers de sortie

Les fichiers commencent par le préfixe `sortie` donné en argument. Les nombres sont écrits avec la culture de la machine.

`sortie_od.txt` : résultats par OD

En-tête : `o;d;temps;cout;ncorr`. Une ligne par OD de la matrice, dans l'ordre des destinations.

Colonne	Contenu
`o`, `d`	Origine et destination
`temps`	Temps moyen attendu (attente comprise). Sans pondération. N'inclut pas `tboa`, sauf option `tboa_dans_temps`
`cout`	Coût généralisé moyen attendu
`ncorr`	Nombre moyen d'embarquements

Une OD en erreur est écrite `o;d;;;` (voir section 8).

`sortie_aff.txt` : affectation par lien

En-tête : `num;i;j;ij;line;volau;boai;alij`. Une ligne par lien dont le volume est strictement positif.

Colonne	Contenu
`num`	Numéro du lien (ordre du fichier réseau, à partir de 0)
`i`, `j`, `ij`, `line`	Nœuds, `i-j`, ligne
`volau`	Volume du lien
`boai`	Embarquements
`alij`	Descentes

`sortie_times.txt` : coûts et temps par lien et destination

En-tête : `d;id;i;j;line;ij;hdwy;time;cout;temps`. Pour chaque destination, une ligne par lien dont le coût moyen est strictement positif : `cout` = `avg_cout` et `temps` = `avg_time` du lien vers cette destination. Ce fichier est toujours écrit et peut être très volumineux (destinations × liens).

`sortie_stg.txt` : stratégies

Écrit seulement si `output_strategies = true`. Le fichier est créé dans tous les cas et reste vide si l'option est `false`. Pour chaque destination : une ligne avec son nom, puis, pour chaque lien :

- une ligne lien : `id;i;j;line;hdwy;time;m;M;M0;n;avg_cout;avg_time;pos;reached` ;
- une ligne par stratégie, commençant par un espace : `line;hdwy;n;p;c;C;pivot`.

Pour les stratégies de transport, la colonne `C` reprend le `M` du lien, commun à toutes ses stratégies (voir section 9).

`sortie_log.txt` : journal des OD en erreur

En-tête : `ligne;raison;contenu`. Voir section 8.

8. Erreurs et journal

OD en erreur. L'OD est écrite `o;d;;;` dans `sortie_od.txt` et une ligne est ajoutée au journal : `ligne;raison;contenu`, avec le numéro de ligne de la matrice, la raison et la ligne d'origine. Le journal est classé par destination, dans l'ordre de traitement. Raisons possibles :

Raison	Cas
demande nulle, négative ou non numérique	`demande ⩽ 0`
origine inconnue du réseau	Le nœud n'existe pas dans le fichier réseau
destination inconnue du réseau	Idem pour la destination
destination inatteignable depuis l'origine	Aucun lien sortant de l'origine n'est atteint depuis la destination

Une OD en erreur ne charge aucune demande.

Erreurs bloquantes. Elles arrêtent le programme avec un message :

- ligne de réseau avec moins de 8 champs, ligne de matrice avec moins de 3 champs (numéro de ligne indiqué) ;
- ligne de type mal formée dans les paramètres, ou type défini deux fois ;
- `cwait ⩽ 0` ;
- erreur dans un thread de calcul.

9. Points d'attention et limites

1. **Séparateur décimal.** Les nombres sont lus et écrits avec la culture régionale de la machine. Sur un Windows en français, la virgule est le séparateur décimal : les fichiers de paramètres et de sortie suivent ce réglage. Garde le même réglage pour écrire et relire les fichiers.
2. **Origine.** Toute la demande d'une OD part du **seul** lien sortant de l'origine qui a le plus petit coût moyen. Il n'y a pas de partage entre plusieurs liens sortants. Pour répartir la demande entre plusieurs lignes, la zone doit avoir un connecteur vers un nœud d'arrêt où se choisissent les stratégies.
3. **Marche de type 2.** La marche traitée comme une ligne à fréquence infinie en milieu de parcours n'est pas implémentée. Un lien à `hdwy = 0` suivi d'une ligne se comporte comme la marche de « type 1 » de l'article (il hérite des stratégies de son pivot). La marche directe vers la destination est, elle, une stratégie à fréquence infinie.
4. **Capacité.** `vcap` est lue mais jamais utilisée. L'affectation est sans capacité.
5. **`C` et `T`.** Ces champs de stratégie ne servent à aucune décision. Dans `sortie_stg.txt`, `C` vaut le `M` du lien pour les stratégies de transport.
6. **Buckets.** L'indice du bucket est `min((coût / alg_parameter)², 10000)`. L'article décrit des intervalles qui dépendent de la racine carrée du temps. L'effet d'un autre exposant n'a pas été testé. `alg_parameter` règle l'ordre de traitement des liens.
7. **Volume des fichiers.** `sortie_times.txt` et `sortie_stg.txt` croissent avec destinations × liens.
8. **Précision.** Les calculs sont en `float` (32 bits).
9. **Temps d'embarquement.** Par défaut, `temps` n'inclut pas `tboa`, comme le temps `T` de l'article. Le coût `cout` l'inclut, via `tboa × cboa`.

10. Validation

Le programme a été comparé aux exemples chiffrés de l'article (poids à 1, `tboa = 0` sauf indication).

Cas	Article	Programme
Exemple 1 : deux lignes à temps égaux	Parts 0,71 / 0,29, T = 24,285	0,714 / 0,286, T = 24,286
Exemple 2 : temps différents	Parts 0,595 / 0,405, T = 22,56	0,595 / 0,405, T = 22,560

Cas	Article	Programme
Exemple 3 : deux lignes et une marche	Parts 0,4 / 0,33 / 0,27	0,4 / 0,333 / 0,267
Réseau Emme, fig. 12 (marche vers E)	T = 24,46, flux 0,143 / 0,286 / 0,175 / 0,111 / 0,169 / 0,117 / 0,571	T = 24,467, mêmes flux à 3 décimales
Réseau Emme, fig. 14 (temps généralisé)	C = 34,84, T = 25,15, mêmes flux à 3 décimales	C = 34,842, T = 25,152, mêmes flux

Pour l'exemple 3, le programme donne T = 16,73 et l'article 16,68. Le calcul à la main avec le bon appariement des temps minimaux donne 16,733.

Autres vérifications, sur des réseaux synthétiques : résultats identiques entre 1, 4 et 8 threads, absence de régression quand aucun type de lien n'est défini, journal vide sur une matrice sans erreur. Aucune validation sur un réseau réel de grande taille n'a été faite.

11. Historique par rapport à la version reçue

Parallélisation. Une destination par unité de travail, une copie du réseau par thread, écriture ordonnée des résultats.

Corrections (elles peuvent changer les résultats) :

1. Les colonnes 7 et 8 du réseau (`allowboai` , `allowalij`) étaient écrasées à 1 à chaque destination. Elles sont respectées.
2. `id_bucket` était passé par valeur : un lien tombant dans un bucket déjà dépassé n'était jamais traité. Il est passé par référence.
3. La restauration d'un lien après une optimisation sans gain n'avait aucun effet. Elle fonctionne.
4. Dans `OptimizeReduce` , le temps du lien et l'embarquement étaient comptés deux fois dans le test de remplacement. Le test compare maintenant `s.c` à la stratégie existante.
5. `pos` est remis à -1 pour chaque destination, ce qui rend le résultat indépendant de l'ordre de traitement.

Sur un jeu de test synthétique (2 112 liens, 14 280 OD), les corrections 2 à 4 changent le coût de 952 OD, dont 940 à la baisse. L'effet de la correction 1 dépend de la présence de restrictions dans le fichier réseau.

Corrections sans effet sur le calcul. Compteur d'avancement correct (la division entière le laissait à 0). Lignes vides finales ignorées. Messages d'erreur avec numéro de ligne. Arrêt avec message si `cwait ⩽ 0` .

Nouveautés. OD en erreur (ligne vide dans `_od`, fichier `_log`). Poids `cmap` et `cboa` par type de lien. Option `tboa_dans_temps`. Argument `nb_threads`. Lignes de commentaire et lignes vides acceptées dans le fichier de paramètres.

Essayé puis retiré. Ajouter `tboa` au temps par défaut : l'article publie un temps sans temps d'embarquement. L'option `tboa_dans_temps` permet de l'ajouter.

12. Exemple complet

Exemple 2 de l'article : deux lignes entre un arrêt et la destination, avec des temps et des intervalles différents.

`reseau.csv`

```
Z;S;C;0;0;100;1;1
S;D;L1;20;12;100;1;1
S;D;L2;15;30;100;1;1
```

`matrice.csv`

```
Z;D;1
```

`parametres.txt` (poids à 1, pas d'embarquement)

```
1
1
0
0
true
10
```

Commande :

```
Mintc reseau.csv matrice.csv res parametres.txt
```

`res_od.txt`

```
o;d;temps;cout;ncorr
Z;D;22.55952;22.55952;0.9999999
```

`res_aff.txt` (extrait)

```
num;i;j;ij;line;volau;boai;alij
1;S;D;S-D;L1;0.595238;...
2;S;D;S-D;L2;0.4047619;...
```

La ligne 1 reçoit 59,5 % de la demande et la ligne 2 40,5 %. La ligne la plus rapide, malgré son intervalle de 30 minutes, reçoit une part supérieure à celle qu'aurait donnée une répartition par fréquences seules (28,6 %).