

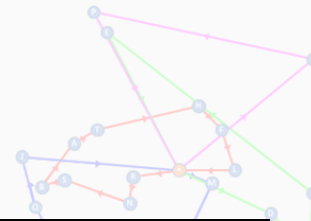
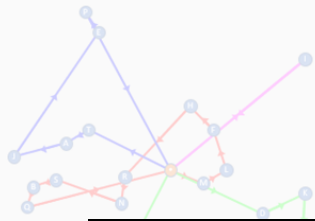
Tournées - Plus Proche Voisin			
Camion	Charge	Distance	Tour
1	249	85	dep-M-L-F-H-R-N-S-B-Q-dep
2	205	95	dep-T-A-J-E-P-dep
3	180	178	dep-D-K-C-G-O-dep
4	20	104	dep-I-dep

Tournées - Ecartements			
Camion	Charge	Distance	Tour
1	186	100	dep-G-O-Q-B-J-dep
2	236	186	dep-C-K-D-S-A-T-dep
3	73	181	dep-I-P-H-dep
4	160	167	dep-M-R-E-F-L-M-dep

Tournées - Recuit Simulé			
Camion	Charge	Distance	Tour
1	200	143	dep-R-N-S-B-A-T-H-F-L-dep
2	231	160	dep-M-G-O-Q-J-dep
3	159	188	dep-D-C-K-E-dep
4	45	173	dep-I-P-dep

## Transport Management Dashboard Project

### Manuel du developpeur



MONTANARI Filippo

Ce document contient les aspects techniques principaux du Transport Management Dashboard.

En développant ce projet, nous avons adopté autant que possible une approche modulaire et d'optimisation computationnelle. L'utilisation des structures (matrices) a été privilégiée pour l'enregistrement et l'élaboration des données, ce qui accélère énormément le déroulement des algorithmes. En outre, afin de limiter l'espace requis en mémoire lors du lancement des différentes procédures, le protocole de passage des arguments *ByRef* a été largement utilisé.

Le résultat est un outil qui peut appliquer les trois méthodes de séquençement à un grand nombre de villes, et dont les limites sont de type graphique bien avant de rencontrer des limites à niveau computationnelle.

## Tableau des matières

Ajouter des villes .....	2
Effacer des villes .....	3
Lancer une méthode de résolution .....	4
Procédure principale de gestion des méthodes .....	5
Méthode du Plus Proche Voisin .....	7
Méthode des Ecartements .....	7
Méthode du Recuit Simulé .....	9
Postcheck du dépassement des contraintes .....	11
Ecriture des résultats dans la feuille .....	11
Représentation graphique des tournées .....	12
Procédures d'effacement .....	14
Faire des Comparaisons .....	14

## Ajouter des villes

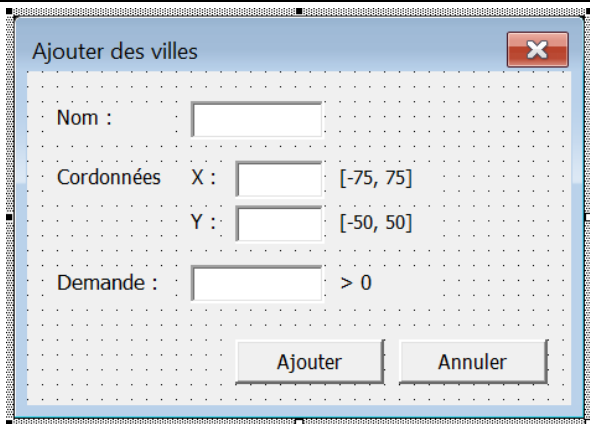
### Aperçu

Emplacement Feuilles > AddCities

Ce formulaire permet à l'utilisateur d'ajouter une ville au tableau des données.

En cliquant « Ajouter », le code vérifie si X, Y et la demande sont numériques et s'ils respectent les conditions écrites à côté. Une demande  $\leq 0$  n'est pas admissible, tandis que des coordonnées hors des limites spécifiées ne posent que des problèmes à niveau graphique.

Si les données saisies sont valides, la nouvelle ville est attachée au fond du tableau des villes.



### Pseudo-code

```
Sub AddVilles_Click()
```

Enregistrement des valeurs du formulaire dans des variables locales

```
If X, Y and Demande sont numeriques et Demande > 0 Then
```

```
    If la valeur de X ou Y sort des limites spécifiés Then
```

```
        MsgBox pour demander si l'utilisateur souhaite toujours ajouter la ville
```

```
        If reponse = Yes Then ajoute = True
```

```
    End If
```

```
    If ajoute Then
```

```
        compter le nombre de villes dans le tableau des données
```

```
        Ajouter les informations saisies dans la première ligne libre
```

```
        Vider les variables
```

```
        'mise en forme de la nouvelle ligne
```

```
        FormatCityLine n
```

```
    End If
```

```
Else
```

```
    MsgBox pour informer l'utilisateur que les valeurs ne sont pas valides
```

```
End If
```

```
End Sub
```

### Procédures attachées

Nom	Emplacement	Description
OpenAddCities()	Modules > F_Extra	Ouverture du formulaire
FormatLastCityLine(n)	Feuilles > F_Extra	Mise en forme de la ligne de la ville ajoutée
CloseForm_Click()	Feuilles > AddCities	Fermeture du formulaire

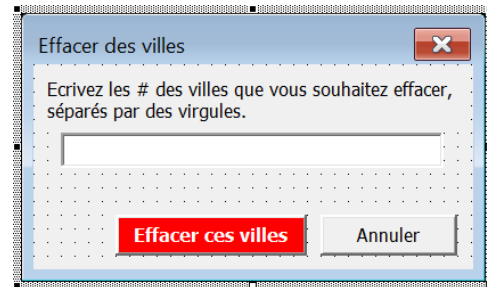
## Effacer des villes

### Aperçu

Emplacement Feuilles > RemoveCities

Ce formulaire permet à l'utilisateur d'effacer des villes selon leur # dans le tableau.

Les # des villes à effacer doivent être séparées par des virgules. Le code reconnaît et gère les entrées non numériques, et les doublons en les supprimant.



### Pseudo-code

```
Sub Confirm_Click()

If le string saisi par l'utilisateur n'est pas vide Then

    Convertir le string avec les # des villes en array avec la fonction Split()
    \ "2,1,3" -> ["2"]["1"]["3"]

    Trier les # des villes et convertir en format Long
    If il y a des # non numériques Then giveError = True et ignorer ce #

    While il a des # des villes valides encore à effacer
        If une # a déjà été rencontré Then
            Ignorer ce # et giveError = True
        Else
            Supprimer la ligne de la ville #
        End If
    Wend

    If giveError = True
        MsgBox qui communique la non-conformité de certains # saisis
    End If

End If

End Sub
```

### Procédures attachées

Nom	Emplacement	Description
OpenRemoveCities()	Modules > F_Extra	Ouverture du formulaire
FormatLastCityLine(n)	Feuilles > F_Extra	Mise en forme de la dernière ligne
CloseForm_Click()	Feuilles > RemoveCities	Fermeture du formulaire

## Lancer une méthode de résolution

### Aperçu

#### Emplacement

Feuilles > AddCities

Ce formulaire permet de lancer la procédure Main (ou MainComparaison, selon le cas) après avoir spécifié :

- la méthode de résolution souhaitée ou le « mode comparaison »,
- la contrainte de charge,
- la contrainte de distance,
- le choix d’affichage du distancier.

**Méthodes de résolution**

Sélectionnez une méthode de résolution :

☒ Plus proche voisin    ☐ Ecartements    ☐ Recuit simulé

☐ Comparer les trois méthodes

Saisissez les contraintes de capacité et distance :

Laissez vide la zone de saisie si vous ne voulez pas ajouter la contrainte associée.

Max charge :     Max distance :     nombre entiers majeur de 0.

☒ Afficher le distancier

**Lancer**    **Annuler**

### Pseudo-code

```
Sub Launch_Click()  
  Enregistrement de max charge, max distance et choix d’affichage du distancier  
  If l’utilisateur a choisi le mode comparaison Then  
  
    Fermer le formulaire  
    Lancer MainComparaison <- charge, distance, choixDistancier  
  
  Else  
  
    Enregistrement de la méthode de résolution choisie  
    Fermer le formulaire  
    Lancer Main <- method, charge, distance, choixDistancier  
  
  End If  
End Sub
```

### Procédures attachées

Nom	Emplacement	Description
OpenLaunchSolution()	Modules > A_MainModule	Ouverture du formulaire
CloseForm_Click()	Feuilles > LaunchSolution	Fermeture du formulaire

## Procédure principale de gestion des méthodes

### Aperçu

**Emplacement** Modules > A\_MainModule

Cette procédure est lancée par le formulaire « Lancer une méthode de résolution ».

Elle exécute en séquence :

1. le compte du nombre de villes à séquencer,
2. l'enregistrement du tableau des données,
3. le calcul, l'enregistrement et le traçage (si requis) du distancier,
4. l'initialisation de la matrice du parcours,
5. le calcul les écartements si la méthode choisie est celle des écartements,
6. la validation des contraintes, qui doivent être numériques et > 0.

Ensuite, pour chaque tournée (portion du parcours qui respect les contraintes) :

1. lancement de la méthode de résolution sélectionnée jusqu'à l'atteint des contraintes,
2. enlèvement des villes de la tournée à partir de la dernière afin de respecter les contraintes,
3. écriture des résultats de la tournée du camion actuel dans la colonne « Ordre » et le tableau « Tournées ».

Cette procédure est le cœur de l'outil et la connexion entre toutes les procédures qui participent à la résolution du problème. Pour cette raison, un bon degré de standardisation a été nécessaire, ainsi que l'utilisation massive de arguments [ByRef](#) afin de simplifier la décomposition du code en plusieurs procédures et limitant au même temps l'espace requis en mémoire.

### Pseudo-code

**Sub** Main(method, MaxCapacite, MaxDistance, TraceDistancier)

Compte du nombre de villes n dans le tableau

Enregistrement des données dans la matrice DONNEES

Calcul des distances entre villes et enregistrement dans la matrice DISTANC

**If** TraceDistancier **Then**

Tracement du distancier à droite du tableau des tournées

Mise en forme du distancier

**End If**

Initialisation de la matrice PARCOURS :

-1 dans toutes les ligne des colonnes de 1 à n

0 dans la colonne n+1, qui contient le nombre de villes contenues dans la ligne

PARCOURS							
	1	2	3	..	..	n	n+1
1	-1	-1	-1	..	..	-1	0
2	-1	-1				-1	0
:	:					:	:
n/2	-1	..	..	..	..	-1	0

**If** method = "Ecartements" **Then**

Calcul des écartements et enregistrement dans la matrice ECART

**End If**

**If** method = "Ecartements" **Then**

Paramétrage de la méthode du recuit simulé :  $\tau = 30000 + \sqrt{n * 900}$

**End If**

Vérification des valeurs max de capacité et distance :

**Function** CheckContrainte(valeur, ContrainteName, limite = 0) **As** Long

**If** valeur <> "" et (non-numérique ou <= 0) **Then**

valid = **False** et message à l'utilisateur

**ElseIf** valeur = "" **Then** 'contrainte ne pas saisie

valid = **False**

**End If**

**If** **Not** valid **Then**

```

        CheckContrainte = -1          'la fonction retourne -1
    Else
        CheckContrainte = CLng(valeur) 'la fonction retourne la valeur
        End If                        de la contrainte en format Long
    End Function

```

```

LastWrittenIndex = 0 'indice dans PARCOURS de la dernière ville écrite
CamionN = 1          'numéro de camion
Finish = False       'variable booléenne qui définit la fin de la procédure
MsgContrTooSmall = False

```

**While Not** Finish

Mise à 0 des compteurs de capacité et distance

**If** ils restent plus que 2 villes à placer **Then**

Lancement d'une instance de la méthode sélectionnée jusqu'au placement de toutes les villes ou au dépassement d'une contrainte

**Else**

Lancement de la méthode qui place les 2(1) dernière(s) ville(s)

's'ils ne restent que 1 ou 2 villes à placer il vaut mieux le faire avec cette méthode très vite et simple par rapport aux autres

```

Sub Last2Villes(n, ByRef PARCOURS, ByRef DONNEES, ByRef DISTANC, ByRef
Capacite, ByRef Distance)

```

**For** tout ville restante faire

recherche d'un # de ville qui n'a pas encore été placé

placement dans la première place libre dans PARCOURS

mise à jour du nombre de villes dans PARCOURS

calcul de la capacité de la tournée

**Next** ville

Calcul de la distance de la tournée (composée pas 1 ou 2 villes)

**End Sub**

**End If**

Post check de la tournée pour lui enlever des villes tant que les contraintes restent dépassées

**If** method = "Ecartements" **Then**

Clôture de tous les écartements impliquant les villes de la tournée qui vient d'être formée

**End If**

Ecriture des résultats dans la feuille

'mise à jour des valeurs d'itération et contrôle de la condition de sortie

CamionN = CamionN + 1

Mise à jour de LastWrittenIndex

**If** LastWrittenIndex = n **Then**

Finish = True

**End If**

**Wend**

Mise en forme de l'entête du tableau des tournées

**End Sub**

### Procédures attachées

Nom	Emplacement	Description
DistancierLayout()	Modules > F_Extra	Mise en forme du distancier
CheckContrainte(..)	Modules > A_MainModule	Vérifie la conformité des contraintes
Last2Villes(..)	Modules > B_SolvingMethods	Séquence les 2(1) dernière(s) ville(s)

## Méthode du Plus Proche Voisin

### Aperçu

Cette procédure trouve une séquence de passage en utilisant la méthode du plus proche voisin. Elle se termine lorsque toutes les villes ont été placée ou au dépassement des contraintes de charge ou distance.

### Pseudo-code

```
Sub PlusProcheVoisin(n, ByRef PARCOURS, ByRef DONNEES, ByRef DISTANC, ByRef Capacite, ByRef Distance, MaxCapacite, MaxDistance)
```

A chaque instance, on re(part) toujours du dépôt pour la recherche de la première ville de la tournée

```
While Not ContrainteAtteinte And il y a des villes à visiter  
    Set ContrainteAtteinte = False
```

On cherche la ville la plus proche à la DernièreVilleVisitée entre celles qui n'ont pas encore été visitées

Mis à jour des compteurs des contraintes avec les infos de la ville choisie

```
If limite de capacité dépassée Then
```

```
    Set ContrainteAtteinte = True
```

```
End If
```

```
DistanceWithDepot = Distance + distance entre la dernière ville et le dépôt
```

```
If DistanceWithDepot a dépassé la distance limite Then
```

```
    Set ContrainteAtteinte = True
```

```
End If
```

Mis à jour du nombre de villes dans le parcours

Ajoute de la ville trouvée pendant cette itération au fond du parcours

La ville qui vient d'être visité devient la DernièreVilleVisitée

```
Wend
```

Si on a atteint les contraintes on ajoute le retour vers le dépôt au compteur de distance

```
End Sub
```

## Méthode des Ecartements

### Aperçu

Emplacement	Modules > B_SolvingMethods
-------------	----------------------------

Cette procédure trouve une séquence de passage en utilisant la méthode des écartements. L'introduction des contraintes de capacité et distance a compliqué énormément la mise en place de cette algorithme, vu que des contrôles et des fusions de parcours se sont rendus nécessaires à l'ajoute de chaque nouvelle ville. Ceci fait que la méthode puisse résulter lourde est ralentie lorsqu'elle est appelée à séquencer un grand nombre de villes sous contraintes. Par contre, sans contraintes la procédure est moins lourde en termes de complexité algorithmique.

## Pseudo-code

```
Sub Ecartements(n, n_ecart, ByRef PARCOURS, ByRef ECART, DONNEES, DISTANC, MaxCapa-
cite, MaxDistance)

Set Finish = False
While Not Finish
    Set found = False

    '1) RECHERCHE DU BINOME AVEC LE MEILLEUR ECARTEMENT
    While Not found
        Set found = True
        On cherche le binôme avec l'écartement maximum entre ceux qui n'ont pas en-
        core été choisis
        On clôture le binôme ville1-ville2 dans la matrice ECART

        '2) VERIFICATION DE LA NON-CREATION DE FOURCHES ET/OU BOUCLES
        If ville1 ou ville2 est contenue dans un parcours déjà enregistré
            bifurcation -> Set found = False
        End If
        If ville1 et ville2 sont les extrêmes d'un parcours déjà enregistré
            boucle -> Set found = False
        End If
    Wend

    '3) ECRITURE DU BINOME DANS LA MATRICE PARCOURS
    Set placed = False
    While Not placed And il y des parcours à inspecté dans PARCOURS

        If le parcours inspecté a des villes à l'intérieur Then
            If le binôme ville1-ville2 peut être attaché au parcours Then
                On décale le parcours si nécessaire
                On inverse le binôme -> ville2-ville1, si nécessaire
                On attache le binôme en tête ou en queue au parcours
                Set placed = True
                Mis à jour des compteurs de capacité et distance sans compter le dépôt
            End If
        End If

        If le binôme n'a pas pu être attaché à aucun parcours existant Then
            On place le binôme dans la première ligne libre
            Set placed = True
            Mis à jour des compteurs de capacité et distance sans compter le dépôt
        End If
    Wend

    '4) FUSION DES PARCOURS EXISTANTS
    'boucle primaire, indice nPP : parcours auquel on attachera le parcours secon-
    daire
    While il y a encore des lignes dans la matrice PARCOURS
        If il y a un parcours dans la ligne nPP Then

            'boucle secondaire, indice nPS : recherche du parcours à attacher
            While il y a encore des lignes dans la matrice PARCOURS
                Set moved = False
                If il y a une correspondance tête-tête, tête-queue, queue-tête ou
                queue-queue entre les parcours aux lignes nPP et nPS Then
                    On décale le parcours primaire si nécessaire
                    On inverse le parcours secondaire si nécessaire
                    On attache le parcours secondaire au primaire
                    Set moved = True
                    Mis à jour du compteur de capacité, le compteur de distance ne néces-
                    site pas de modifications
                End If

                If moved Then
```



On efface le parcours secondaire de la matrice PARCOURS  
On reset l'indice des parcours primaire nPP = 0

Cette dernière action est nécessaire puisque la fusion des deux parcours peut avoir donné lieu à l'apparition de nouvelles coïncidences dans les lignes déjà parcourues.

```
End If
Wend

End If
Wend

'5) VERIFICATION DES CONDITIONS DE SORTIE DE LA METHODE
If on a placé toutes les n villes dans le parcours Then
    Finish = True
End If

If on a plus d'un parcours dans PARCOURS Then
    On évalue la distance résultante de la fusion des parcours en ajoutant aussi la distance de/vers le dépôt. Les parcours sont fusionnés chacun avec le suivant, et la direction de fusion (tête / queue) est celle qui minimise la distance. On utilisera la distance totale résultante pour le contrôle qui suit.
End If
If le parcours fusionné dépassé l'une des contraintes Then
    On écrit le parcours fusionné dans la première ligne de PARCOURS et on réinitialise les autres lignes
    Finish = True
End If

Wend

End Sub
```

## Méthode du Recuit Simulé

### Aperçu

Emplacement Modules > B\_SolvingMethods

Cette procédure trouve une séquence de passage en utilisant la méthode métaheuristique du Recuit Simulé. L'algorithme consiste à sélectionner deux villes par hasard et à les échanger : si la distance du nouveau parcours est inférieure à celle du précédent, l'échange est retenu, sinon l'échange n'est retenu qu'avec une certaine probabilité qui diminue avec l'avancement de l'algorithme.

L'avantage de cette méthode est visible avec des grosses quantités de villes et des chaînes de livraison longues. En effet, la complexité algorithmique est d'ordre  $O(n\sqrt{n}) + O(\tau)$ , tandis que la méthode du plus proche chemin est d'ordre  $O(n^3)$  et pour celle des écartements c'est  $O(n^4)$ .

Comme pour la méthode des écartements, dans certains cas l'introduction des contraintes de capacité et distance peut impacter la bonté absolue des résultats.

### Pseudo-code

```
Sub RecuitSimule(n, LastWrittenIndex, ByRef tau, ByRef PARCOURS, ByRef DONNEES, ByRef DISTANC, ByRef Capacite, ByRef Distance, MaxCapacite, MaxDistance)
```

NB :

1) la solution initiale fournie à cette méthode (dans la matrice ByRef PARCOURS) est calculée avec la méthode du plus proche voisin ;

2) la valeur de tau, qui définit la vitesse du refroidissement, est calculé selon la formule  $\tau = 30000 + \sqrt{n \cdot 900}$ , ce qui donne  $\tau = 30000$  avec 0 villes et  $\tau = 60000$  avec 10000 villes.

Définition des paramètres propres de la méthode du Recuit Simulé :

T0 -> Température initiale

Tmin -> Température minimale à atteindre avec le refroidissement

Energie totale de l'état (parcours) = sa distance

**While** le paramètre « Température » est majeur de Tmin

On choisit les deux villes à échanger :

- la première **i** est choisi entre celles dans PARCOURS, hormis celles qui ont déjà été écrites dans la feuille dans le tableau des résultats
- la deuxième **j** peut être n'importe quelle ville, sauf (comme pour la 1ère) celles qui ont déjà été écrites dans la feuille dans le tableau des résultats

**While i = j** <-> les deux villes coïncident **Then**

Rechoisit la deuxième ville

**End If**

Création de la fluctuation d'état (ordre des villes) et calcul de la nouvelle énergie (distance) :

**Sub** Fluctuation(n, **i**, **j**, ByRef LastWrittenIndex, ByRef PARCOURS, ByRef DONNEES, ByRef DISTANC, ByRef Capacite, ByRef Distance)

Inversion des villes # **i** et **j** et recalcul ciblé de la capacité et la distance du parcours.

Le fait de faire des calculs ciblés pour mettre à jour les compteurs des contraintes permet de limiter la complexité algorithmique. En effet, l'alternative serait une boucle For qui reparcourt la tournée entière et recalcule la charge et la distance à partir de 0 -> complexité ~n

**End Sub**

Energie finale = distance après la fluctuation

Algorithme de Metropolis pour évaluer si retenir ou pas la fluctuation :

**Sub** Metropolis(n, ByRef EnergInit, EnergFin, T, **i**, **j**, ByRef LastWrittenIndex, ByRef PARCOURS, ByRef DONNEES, ByRef DISTANC, ByRef Capacite, ByRef Distance)

**If** l'énergie final est mineur de l'énergie initiale **Then**

énergie du nouvel état = énergie système, et on retient le nouveau parcours

**Else**

dE = EnFin - EnInit

**If**  $\text{Rnd}(0-1) > \exp\left(-\frac{dE}{T}\right)$  **Then**  
la fluctuation est retenue

**Else**

retour au parcours antérieur :

**Sub** Fluctuation(..)

re-inversion des villes **i** et **j** et recalcul de capacité et distance

**End Sub**

**End If**

**End If**

**End Sub**

Application de la loi de refroidissement :

temps = temps + 1

Température =  $T0 * \exp\left(-\frac{\text{temps}}{\tau}\right)$

Composante probabiliste de la méthode du recuit simulé. Plus la température décroît, moins ça sera probable de retenir une fluctuation désavantageuse.

<b>Wend</b>		
<b>End Sub</b>		
<b>Procédures attachées</b>		
Nom	Emplacement	Description
Fluctuation(..)	Modules > B_SolvingMethods	Inversion de deux villes aléatoires
Metropolis(..)	Modules > B_SolvingMethods	Evaluation statistique pour choisir si retenir ou écarter la fluctuation

## Postcheck du dépassement des contraintes

<b>Aperçu</b>	
<b>Emplacement</b>	Feuilles > C_PostCheckContraintes
<p>Comme on l’a pu peut-être remarquer, les méthodes de résolution donnent des parcours où les contraintes de capacité et distance sont en fait dépassées. La raison est que l’intégration de cette petite procédure dans les méthodes a été jugée très compliqué, donc nous avons choisi de faire une simplification en ajoutant cette vérification à posteriori des contraintes, standard pour les trois méthodes.</p>	
<b>Pseudo-code</b>	
<pre> <b>Sub</b> PostCheck(n, ByRef LastWrittenIndex, ByRef PARCOURS, ByRef DONNEES, ByRef DISTANC, ByRef Capacite, ByRef Distance, MaxCapacite, MaxDistance, ByRef MsgContrTooSmall)  <b>While</b> au moins l’une des contrainte reste dépassé <b>And Not</b> ContrTooSmall   <b>If</b> dans la tournée ne reste qu’une seule ville <b>Then</b>     Set ContrTooSmall = <b>True</b>     Set MsgContrTooSmall = <b>True</b>   <b>Else</b>     On enlève la dernière ville de la tournée de PARCOURS, en mettant aussi à jour le nombre de ville dans la ligne (--1)     On met à jour les compteurs de charge et distance   <b>End If</b> <b>Wend</b>  La variable MsgContrTooSmall, si <b>True</b>, est utilisée ensuite par Main(..) pour lancer à l’utilisateur le message que certaines contraintes n’ont pas pu être respectée car trop restrictives.  <b>End Sub</b> </pre>	

## Ecriture des résultats dans la feuille

<b>Aperçu</b>	
<b>Emplacement</b>	Modules > D_WriteResults
<p>Cette procédure écrit les résultats de la tournée actuelle dans la colonne « Ordre » – avec le format <b>&lt;#Camion&gt;</b> (&lt;ordre de passage&gt;) – et dans le tableau « Tournée(s) »</p>	
<b>Pseudo-code</b>	

```

Sub WritingResults(n, LastWrittenIndex, CamionN, ByRef PARCOURS, ByRef Capacite,
ByRef Distance, ByRef DONNEES)

Ecriture de #Camion, charge et distance de la tournée dans le tableau « Tournée(s) »

For toute ville dans le parcours
    Ecriture de #Camion et #ville dans la colonne « Ordre » à la ligne correspondante
    Ajoute du nom de la ville au fond de la ligne de la tournée dans le tableau des
    tournées
Next ville

End Sub

```

## Représentation graphique des tournées

### Aperçu

**Emplacement** Modules > E\_TraceOnMap

Cette procédure lance dans l'ordre :

- 1) l'effacement de tout ovale et ligne dans la feuille
- 2) la procédure qui trace les lignes pour la tournée
- 3) la procédure qui dessine les cercles pour la tournée

### Pseudo-code

```

Sub Tracer()

debutX = 660
debutY = 367

ClearOvalsAndLines
If il y a des résultats dans le tableau des données Then

    Sub TraceConnectors(debutX, debutY)

        Compte du nombre de villes et du nombre de camions en se basant sur la colonne
        « Ordre »

        Enregistrement des coordonnées du dépôt dans la 1ère ligne de la matrice CORD

        For #ville de 1 a n
            Enregistrement du #Camion dans la colonne 1 de la matrice AUX
            Enregistrement l'ordre de passage dans la colonne 2 de la matrice AUX

            Enregistrement des coordonnées X, Y de la ville numéro # dans CORD
        Next ville

        CREATION DE LA MATRICE DES CONNEXIONS :
        on crée la matrice CONNEXIONS à partir des informations dans la matrice AUX
        la matrice CONNEXIONS suivra le schème :
            line_index | col0      col1      col2
            #passage   | #n°camion #n°ville_1 #n°ville_2

        For tout Camion i
            #passage = 1
            While il y a des autres villes à parcourir par le camion actuel
                k = 1 (ligne prise en compte dans la matrice AUX(#camion|#passage))
                While la prochaine ville du camion n'est pas trouvée

```

```

If il y encore des lignes à parcourir dans AUX Then
  If i et #passage correspondent aux données à la ligne k de AUX Then
    CONNEXIONS(Line,0) = i <- #Camion
    CONNEXIONS(Line,2) = k <- #ville
    CONNEXIONS(Line+1,1) = k <- #ville
    Next Line dans CONNEXIONS
  End If
Else
  Le camion i peut rentrer au dépôt
  CONNEXIONS(Line,0) = i <- #Camion
  CONNEXIONS(Line,2) = 0
  Next Line dans CONNEXIONS
End If
Prochaine ligne k
Wend
Prochaine #passage
Wend
Next Camion i

For chaque connexion dans la matrice CONNEXIONS
  On extrait x1, y1, x2, y2 et on calcule le n°couleur du camion
  Traçage du segment de connexion
  Sub Segments(x1, y1, x2, y2, debutX, debutY, color_number)

  Conversion coordonnées arc ville1-ville2 -> coordonnées feuille Excel
  Création d'une ligne de (x1,y1) à (x2,y2), de la couleur spécifiée
  Création d'une deuxième ligne superposée, avec la moitié de la longueur
  par rapport à la première et une flèche à l'extrémité.
  'la présence de cette flèche ne vise qu'à donner une meilleure lisibilité
  On vide les variables x1, y1, x2, y2, color_number

  End Sub
Next connexion
End Sub

```

**End If**

```

Sub TraceVilles(debutX, debutY)
For chaque ville dans le tableau des données
  Sub Cercle(X, Y, lettre, demande, debutX, debutY, Optional mini)
  Conversion coordonnées ville -> coordonnées feuille Excel
  Création dans la feuille d'un ovale en position X, Y
  Mise en forme du cercle selon son rôle et selon la variable mini
  End Sub
Next ville
End Sub

End Sub

```

### Procédures attachées

Nom	Emplacement	Description
TraceVilles(..)	Modules > E_TraceOnMap	Lance Cercle(..) pour chaque ville
Cercle(..)	Modules > E_TraceOnMap	Création et positionnement d'un ovale
TraceConnectors(..)	Modules > E_TraceOnMap	Lance Segments (..) pour chaque arc
Segments(..)	Modules > E_TraceOnMap	Création et positionnement d'une ligne
TraceQueLesVilles()	Modules > E_TraceOnMap	Lance Cercle(..) pour chaque ville et peut être lancé de façon indépendante

## Procédures d'effacement

Nom	Emplacement	Description
ClearAll()	Modules > F_Extra	Efface tout sauf les données d'input.
EffacerLaDerniere-Ville()	Modules > F_Extra	Supprime la dernière ligne du tableau des données.
ClearColonneOrdre(..)	Modules > F_Extra	Efface la colonne « Ordre ».
ClearTableauTournee()	Modules > F_Extra	Efface le tableau des tournées.
ClearOvalsAndLines()	Modules > F_Extra	Efface tout ovale et toute ligne dans la feuille.
ClearDistancier(..)	Modules > F_Extra	Efface le distancier sur la feuille.

## Faire des Comparaisons

Aperçu

Emplacement	Modules > G_MakeComparisons
-------------	-----------------------------

Cette fonctionnalité, ajoutée en dernier, permet de lancer en séquence les trois méthodes et afficher les résultats l'un à côté de l'autre dans une feuille créée à cet effet.

La procédure reprend tous les éléments vus dans les procédures ci-dessus, modifiées à fur et a mesure pour tenir en compte les décalages de position des données et des objets.

Procédures attachées

Nom	Emplacement	Description
FormatSheet_FaireDesComparaisons()	Modules > G_...	Création et mise en forme de la feuille « Faire des Comparaison ».
DeleteSheet_FaireDesComparaisons()	Modules > G_...	Suppression de la feuille « Faire des comparaisons ».