

Discrimination des surfaces par distance de débardage et par analyse de pentes

Killian BAUE, Agnès DAVIERE, Louise DUBOST, Elda PERONNET

2024-09-13

Contents

Présentation du projet	2
Méthode	2
Choix de la zone d'étude	2
Recherche et caractérisation de la desserte	2
Données issues de OpenStreetMap	3
Données issues de l'IGN	5
Matérialisation des distances de débardage autour de la desserte	8
Création de buffers	8
Rasterisation des buffers	9
Calcul de la pente à partir du MNT	10
Pente avec R	10
Calcul des pentes avec QGis et comparaison des deux méthodes	12
Création de la carte d'exploitabilité	12
Résultats	14
Carte de desserte	14
Cartes des buffers autours de la desserte	14
Carte catégories de pentes	16
Carte d'exploitabilité	17
Les limites et perspectives	18
L'optimisation des calculs de pente	18
Optimisation de la carte d'exploitabilité	18
Perspective desserte	19

Présentation du projet

Identifier les zones les plus accessibles et les plus adaptées aux opérations de débardage est essentiel pour les gestionnaires forestiers. Ce projet a pour objectif d'automatiser la détermination de catégorie de surface selon la distance de débardage et la pente sur une zone connue. La distance de débardage par rapport à la desserte est classée selon 3 catégories : 20 – 50 – 100m (IGN 2023). La couche de desserte et le MNT sont récupérés à partir du package **happign** (Carteron 2024) de R. Le traitement a ensuite été réalisé en parallèle avec Rstudio et QGis afin de mener une étude comparative.

Ce projet a abouti à la création d'une carte d'exploitabilité couplant le relief et les distances de débardage par rapport aux pistes. Nous avons atteint les objectifs du projet en fournissant un outil permettant d'optimiser l'exploitation forestière et faciliter la prise de décision.

Méthode

Choix de la zone d'étude

La zone d'étude a été choisie de manière à respecter plusieurs critères :

- comporter des zones forestières
- présenter une topographie avec du relief
- posséder un réseau de desserte d'une taille suffisante avec différentes natures de routes
- couverte par la technologie Lidar. Un axe d'étude étant la comparaison de différentes sources de MNT, il est judicieux de choisir une zone d'étude ayant déjà fait l'objet d'une campagne Lidar et dont les données retraitées sont disponibles.
- une superficie raisonnable pour optimiser les temps de calcul

La commune d'étude a été choisie suite à des validations des critères précédents est la commune de Saxon en Haute-Savoie. Saxon est une commune de 500 habitants recouverte à plus de 60 % par des zones forestières. Lien wikipedia : <https://fr.wikipedia.org/wiki/Saxon>

L'obtention du cadastre de la commune se fait via la fonction `get_apicarto_cadastre()` du package **happign**.

Ci-dessous, la fonction permettant l'import des données cadastrales :

```
importer.cadastre <- function(insee_code) {  
  # importation des parcelles cadastrales avec le code insee  
  get_apicarto_cadastre(  
    x = insee_code,                      # code insee de la commune  
    type = "parcelle",                   # type de donnees, ici 'parcelle'  
    source = "PCI",                     # source des donnees 'PCI'  
  )  
}
```

Les données cadastrales sont re-projetées en système de coordonnées Lambert 93 (EPSG:2154). Cette re-projection permet que toutes les données géographiques soient alignées dans le même système de référence.

Recherche et caractérisation de la desserte

Pour obtenir des données de dessertes viables, deux sources de données ont été explorées : les données de l'IGN et les données d'OpenStreetMap.

Données issues de OpenStreetMap

Pour utiliser les données d'OSM (OpenStreetMap), le package `OSM_dataaa` été utilisé.

Deux fonctions ont été utilisées pour explorer et récupérer les données OpenStreetMap. La première fonction, `explore.osm.keys`, récupère toutes les clés OSM disponibles, les stocke dans un data frame et les affiche dans une fenêtre interactive avec la fonction `View()`. Cela permet une inspection manuelle des clés.

```
# Définir la fonction pour afficher les clés OSM dans une fenêtre interactive
explore.osm_keys <- function() {

  # Étape 1: Récupérer les clés disponibles dans OSM et les stocker dans un data frame
  keys <- available_features()

  # Créer un data frame pour les clés
  keys_df <- data.frame(
    Key = keys)

  # Afficher les premières lignes du data frame pour vérifier
  print(
    head(keys_df))

  # Étape 2: Ouvrir les données dans une fenêtre interactive de RStudio
  View(
    keys_df)

  # Retourner le data frame au cas où l'utilisateur souhaite l'utiliser
  return(
    keys_df)
}
```

La seconde fonction, `get.osm.values`, vérifie si une clé OSM spécifique existe, puis récupère les valeurs associées à partir d'une clé OSM définie en entrée de fonction. Si des données existent, elles sont organisées dans un data frame et affichées pour une analyse plus approfondie.

```
# Définir la fonction qui récupère les valeurs OSM possibles pour une clé spécifique
get.osm.values <- function(osm_key) {

  # Vérifier si la clé existe dans OSM
  keys_available <- available_features()
  if (!(osm_key %in% keys_available)) {
    stop(paste(
      "La clé",
      osm_key,
      "n'existe pas dans OpenStreetMap. Veuillez vérifier la clé."))
  }

  # Étape 1: Récupérer les valeurs possibles pour la clé spécifiée
  values <- available_tags(
    osm_key)

  # Vérifier si des valeurs ont été trouvées pour cette clé
  if (length(values) == 0) {
```

```

stop(paste(
  "Aucune valeur trouvée pour la clé:",
  osm_key))
}

# Étape 2: Créer un data frame pour les valeurs
values_df <- data.frame(
  Value = values)

# Afficher les premières lignes du data frame pour vérification
View(head(
  values_df))

# Étape 3: Retourner le data frame
return(
  values_df)
}

```

Enfin, la fonction `create.osm.map` génère une carte basée sur des données OpenStreetMap pour une zone d'étude spécifique. En entrée, la fonction reçoit les données suivantes : la zone d'étude sous forme de matrice cadastrale contenant diverses informations. Seuls les contours des parcelles sont utilisés dans ce code.

Dans la fonction, la zone d'étude (`zone_etude`) est convertie en une “bounding box” (`bbox`), qui définit les limites géographiques de la requête. Ensuite, une requête OSM est construite pour extraire les données correspondant à une clé (`osm_key`) et une valeur (`osm_value`). Si des lignes OSM sont retournées, elles sont converties au format spatial (`sf`) et ajouté à une carte interactive `leaflet`, avec des polylinnes colorées. Enfin, la carte est centrée sur la zone d'étude et retournée pour affichage.

```

# Fonction pour créer une carte avec les données OSM basées sur une zone d'étude
create.osm.map <- function(zone_etude,
                           osm_key,
                           osm_value = "*") {
  # Convertir zone_etude en bbox
  bbox <- st_bbox(zone_etude)
  bbox_vector <- c(bbox["xmin"],
                  bbox["ymin"],
                  bbox["xmax"],
                  bbox["ymax"])

  # Définir la requête OSM
  opq_bbox <- opq(bbox = bbox_vector) %>%
    add_osm_feature(key = osm_key,
                    value = osm_value)

  # Exécuter la requête et obtenir les données
  cway_zone <- osmdata_sf(opq_bbox)

  # Vérifier si les lignes OSM sont déjà au format sf
  if (!is.null(cway_zone$osm_lines)) {
    if (!inherits(cway_zone$osm_lines, "sf")) {
      cway_zone$osm_lines <- st_as_sf(cway_zone$osm_lines)
    }
  } else {
    message("Aucune donnée OSM pour les lignes n'a été trouvée.")
  }
}

```

```

}

# Créer la carte
map <- leaflet() %>%
  addTiles() %>% # Fond de carte par défaut
  addPolylines(data = cway_zone$osm_lines,
    color = "blue",
    weight = 3,
    opacity = 0.7) %>%
  setView(lng = mean(bbox_vector[c(1, 3)]),
    lat = mean(bbox_vector[c(2, 4)]),
    zoom = 13)

# Afficher la carte
return(map)
}

```

Données issues de l'IGN

Pour obtenir la desserte avec les données de l'IGN, nous utilisons la fonction `importer.routes()` permettant de récupérer les tronçons de route dans une zone tampon spécifiée en utilisant une requête WFS (Web Feature Service). La zone tampon est définie par l'utilisateur, et seuls les tronçons de route interceptant cette zone sont importés, en se basant sur la couche de données “BDTOPO_V3”. Pour s'assurer que toutes les routes, y compris les pistes, de la zone d'étude sont connectées aux routes accessibles aux grumiers, un buffer de 1 km est créé autour des routes importées.

```

## Fonction pour importer les routes ----
importer.routes <- function(zone_tampon) {
  # importation des tronçons de route dans une zone tampon
  get_wfs(zone_tampon,                                # zone tampon pour la requete WFS
    "BDTOPO_V3:troncon_de_route",
    spatial_filter = "intersects"
  )
}

```

La fonction `reprojeter.donnees()` est utilisée pour projeter des données spatiales dans un système de coordonnées spécifique, ici le système Lambert 93. En prenant en entrée les données spatiales et le code du système de coordonnées (CRS), cette fonction applique la transformation nécessaire via la fonction `st_transform()`, garantissant ainsi que les données sont alignées avec le système de coordonnées Lambert 93 utilisé pour l'analyse.

```

## Fonction pour re-projeter les donnees spatiales ----
reprojeter.donnees <- function(donnees,
                                code_crs) {
  # Reprojection des donnees spatiales dans le systeme de coordonnees specifie
  st_transform(donnees, crs = code_crs)
}

```

Une fois les données routières importées, il est nécessaire de vérifier la nature des routes (chemin, sentier, etc.) en consultant la table attributaire du vecteur. La colonne nature dans la table attributaire contient des informations sur le type de chaque tronçon de route, cette analyse permet d'obtenir un aperçu de la répartition des types de routes dans la zone d'étude.

Par la suite, nous définissons que :

- Les routes à une chaussée ainsi que les routes empierrées sont accessibles aux grumiers ;
- Les chemins sont accessibles au skidder / tracteurs forestiers ;
- Les sentiers ne sont pas comptabilisés dans l'analyse bien que ces derniers peuvent être empruntés par les bûcherons en cas d'abatage manuel (ce qui très probablement le cas dans un contexte alpin).

```
## Fonction pour filtrer les routes par type ----
filtrer.routes <- function(routes) {
  # Creation des sous-ensembles de routes selon leur nature
  routes_grumiers <- routes %>%
    filter(nature %in% c("Route à 1 chaussée", "Route empierrée"))

  routes_skidders <- routes %>%
    filter(nature == "Chemin")

  routes_sentiers <- routes %>%
    filter(nature == "Sentier")

  list(
    routes_grumiers = routes_grumiers,
    routes_skidders = routes_skidders,
    routes_sentiers = routes_sentiers
  )
}
```

Cette classification permet de catégoriser les différents types de routes et chemins, facilitant ainsi l'analyse de leur exploitabilité.

L'étape suivante est de pouvoir classifier les différentes routes comme étant exploitables ou non. Dans cet objectif, la fonction `verifier.exploitabilite()` a pour objectif d'évaluer l'exploitabilité des pistes en vérifiant leur connexion directe ou indirecte avec les routes accessibles aux grumiers. La fonction procède selon les étapes suivantes :

- Création d'un buffer autour des routes grumiers : Un buffer de 5 mètres est généré autour des routes accessibles aux grumiers. Ce buffer permet de définir une zone d'influence autour de ces routes ;
- Vérification de la connexion directe : La fonction utilise `st_intersects()` pour déterminer si les pistes (routes skidders) se trouvent dans cette zone tampon. Les pistes connectées directement aux routes grumiers sont identifiées, tandis que celles qui ne le sont pas restent dans une liste distincte ;
- Identification des connexions indirectes : pour les pistes non connectées directement, un processus itératif est utilisé pour vérifier si elles peuvent être connectées indirectement. Un nouveau buffer de 5 mètres est créé autour des pistes déjà identifiées comme exploitables, et les pistes non connectées sont testées à nouveau pour voir si elles se trouvent dans ce buffer. Ce processus se répète jusqu'à ce qu'aucune nouvelle piste exploitée ne soit trouvée ;
- Intersection et extraction des points : une fois toutes les pistes exploitables identifiées, la fonction calcule l'intersection entre ces pistes et les routes accessibles aux grumiers pour déterminer les points de connexion. Si ces intersections ne sont pas des points individuels, elles sont extraites à l'aide de `st_collection_extract()`.

```
## Fonction pour verification de l'exploitabilite des pistes ----
verifier.exploitabilite <- function(routes_skidders,
                                         routes_grumiers,
                                         routes_sentiers,
                                         cadastre_2154)
```

```

{
# Creation d'un buffer de 5 m autour des routes grumiers
buffer_grumiers <- st_buffer(routes_grumiers, dist = 5)

# Verification de la connexion directe entre skidders et grumiers
connexion_directe <- st_intersects(routes_skidders, buffer_grumiers, sparse = FALSE)
skidders_connectes <- routes_skidders[rowSums(connexion_directe) > 0, ]

skidders_non_connectes <- routes_skidders[rowSums(connexion_directe) == 0,]

tous_skidders_exploitables <- skidders_connectes

# Boucle pour trouver les connexions indirectes
repeat {
  buffer_exploitables <- st_buffer(tous_skidders_exploitables,
                                      dist = 5)
  connexion_indirecte <- st_intersects(skidders_non_connectes,
                                         buffer_exploitables,
                                         sparse = FALSE)

  nouveaux_exploitables <- skidders_non_connectes[rowSums(connexion_indirecte) > 0,]

  if (nrow(nouveaux_exploitables) == 0) break

  tous_skidders_exploitables <- rbind(tous_skidders_exploitables,
                                         nouveaux_exploitables)
  skidders_non_connectes <- skidders_non_connectes[rowSums(connexion_indirecte) == 0,]
}

# Intersection entre chemins exploitables et routes grumiers
intersections <- st_intersection(tous_skidders_exploitables, routes_grumiers)
if (!inherits(intersections, "POINT")) {
  place_de_depot <- st_collection_extract(intersections, type = "POINT")
} else {
  place_de_depot <- intersections
}
}

```

Les résultats obtenus sont sauvegardés dans un fichier GeoPackage. Une fonction dédiée `save_layer()` est utilisée pour enregistrer chaque couche de données dans le fichier. Les couches incluses sont :

- Les données cadastrales de la zone d'étude (i.e la commune de Saxel) ;
- Les tronçons de route (desserte globale) ;
- Les sentiers (à titre informatif) ;
- Les pistes exploitables (réutilisée dans la suite du projet) ;
- Les pistes non-exploitables ;
- Les places de dépôt.

```

## Fonction pour sauvegarder une couche dans un fichier geopackage ----
sauvegarder.couche <- function(donnees, chemin_fichier, nom_couche) {
  # Ecriture de la couche dans un geopackage
  st_write(donnees, chemin_fichier, layer = nom_couche, delete_layer = TRUE)
}

```

Matérialisation des distances de débardage autour de la desserte

Création de buffers

Dans cette partie, des distances de débardage autour des pistes ont été modélisées sur le logiciel R uniquement. Dans un premier temps, des buffers ont été délimités autour des pistes accessibles aux grumiers et accessibles aux tracteurs forestiers.

La construction de ces buffers a été réalisée à l'aide de la fonction `zone.buffer`. Cette fonction prend en entrée le dataframe de la desserte contenant un champ contenant les données sur la géométrie des polygones et le système de coordonnées. Elle nécessite également trois distances en mètres à renseigner dans un sens croissant.

- La fonction commence par supprimer les colonnes du jeu de données qui contiennent des valeurs manquantes (NA). Cela garantit que seules les colonnes valides sont conservées pour éviter des erreurs ultérieures dans le traitement spatial.
- Ensuite, la fonction vérifie si les géométries des chemins sont valides, si des géométries sont invalides, la fonction tente de les corriger avec `st_make_valid`.
- Trois zones tampons de différentes distances sont ensuite générées autour des chemins de desserte. Ces distances ont été renseignées au préalable dans les arguments de la fonction. Les buffers ont été créés en utilisant la fonction `st_buffer`, qui entoure chaque chemin d'une zone tampon à la distance définie.
- Pour chaque zone tampon (buffer), une nouvelle colonne appelée `distance` est ajoutée pour indiquer la distance à laquelle chaque buffer a été créé.
- Les trois zones tampons sont ensuite fusionnées dans un seul dataframe. Cela permet d'avoir toutes les zones tampons dans un seul jeu de données, ce qui facilite leur manipulation et leur analyse par la suite.

Enfin, en sortie, la fonction retourne l'objet fusionné contenant les zones tampons autour des chemins de desserte.

```
# Création de zones de buffers autour de la desserte
zone.buffer <- function(
  shape_routes,
  dist_1,
  dist_2,
  dist_3) {

  # Suppression des colonnes avec des valeurs NA
  shape_routes_clean <- shape_routes[, colSums(is.na(shape_routes)) == 0]

  # Vérifier si les géométries sont valides, sinon les corriger
  shape_routes_clean <- st_make_valid(
    shape_routes_clean)

  # Créer des buffers autour des routes avec les distances spécifiées
  buffer_1 <- st_buffer(
    shape_routes_clean,
    dist = dist_1)
  buffer_2 <- st_buffer(
    shape_routes_clean,
    dist = dist_2)
```

```

buffer_3 <- st_buffer(
  shape_routes_clean,
  dist = dist_3)

# Ajouter une colonne "distance" pour identifier chaque zone
buffer_1$distance <- dist_1
buffer_2$distance <- dist_2
buffer_3$distance <- dist_3

# Fusionner les buffers
buffer_merged <- rbind(
  buffer_1,
  buffer_2,
  buffer_3
)

# Retourner l'objet avec les buffers fusionnés
return(buffer_merged)
}

```

Une fois les zones de buffers créées, le dataframe peut être exporté dans un geopackage visualisable et analysable sur le logiciel QGis.

Rasterisation des buffers

Dans un second temps, une fonction permettant de convertir le dataframe contenant les buffers en format raster a été créée. Transformer les données de distance en format raster permet d'effectuer des analyses détaillées à l'échelle des pixels, ce qui est essentiel pour le calcul précis de l'exploitabilité. La fonction **raster.buffer** prend en entrée le dataframe fusionné contenant les zones tampons autour des pistes. Elle prend également en entrée la résolution, le système de coordonnées et les dimensions nrows et ncols.

- La fonction commence par extraire l'étendue spatiale du buffer. Cette étape permet de définir les limites géographiques du raster en fonction des limites du buffer.
- Un raster vide est ensuite créé en utilisant les dimensions spécifiées en entrée de fonction (le système de coordonnées, la résolution , nrows et ncols). C'est dans ce raster que seront implémentés les données issues du dataframe.

-Le buffer est ensuite transformé en raster en utilisant la fonction **rasterize** du package **terra**. La colonne distance du dataframe contenant les buffers est utilisée comme valeur pour les cellules du raster. La fonction de résumé (fun = min) prend pour chaque pixel rencontré la plus petite valeur de distance en cas de chevauchement.

- Enfin, la fonction retourne le raster créé, prêt à être utilisé pour des analyses spatiales ou des visualisations. Le raster de sortie peut-être ensuite exporté en geopackage et utiliser sur d'autres logiciels.

```

# Créer une fonction pour générer un raster à partir d'un buffer avec nrows et ncols en entrée
raster.buffer <- function(
  buffer,
  resolution = 25,
  crs = "EPSG:2154",
  nrows = 300,

```

```

    ncols = 300) {
# Déterminer l'étendue (extent) du buffer
extent_shape <- ext(
  st_bbox(buffer))

# Créer un raster basé sur cette étendue, la résolution, nrows, ncols, et la projection spécifiée
r <- rast(
  nrows = nrows,
  ncols = ncols,
  res = resolution,
  crs = crs,
  extent = extent_shape)

# Rasteriser le buffer en utilisant la colonne 'distance'
buffer_raster <- terra::rasterize(
  x = buffer,
  y = r,
  field = 'distance',
  fun = min)

# Retourner le raster généré
return(buffer_raster)
}

```

Calcul de la pente à partir du MNT

Pente avec R

Cette partie a pour objet le calcul des pentes de la zone d'étude. La démarche est la suivante:

La fonction principale, `get.slope()`, prend en entrée quatre chaînes de caractères,

Cela est illustré dans le script ci-dessous.

```

# Librairies ----
librarian::shelf(happign,terra,tmap,sf)
tmap_mode("view")

# Dossier de travail ----
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# Fonctions temporaires ----

draw <- function(raster,vecteur){
  tm_shape(raster)+
    tm_raster()+
    tm_shape(vecteur)+
    tm_borders("black", lwd = 2)
}

# Fonctions intermédiaires ----

code.insee <- function(code_post, libelle){

```

```

info_com <- get_apicarto_codes_postaux(code_post)
ligne <- which(info_com$libelleAcheminement == libelle)
code_insee <- info_com$codeCommune[[ligne]]
return(code_insee)
}

get.cadastre <- function(code_insee, section, num_parc){
  zone_parca <- get_apicarto_cadastre(code_insee,
                                         type = "parcelle",
                                         code_com = NULL,
                                         section = section,
                                         numero = num_parc,
                                         dTolerance = OL,
                                         source = "PCI"
                                         )
  return(zone_parca)
}

get.mnt <- function(zone_parca){
  mnt_layer_name <- "ELEVATION.ELEVATIONGRIDCOVERAGE"
  zone_parca_buffered <- st_buffer(zone_parca, dist = 100)
  mnt <- get_wms_raster(x = zone_parca_buffered,
                         layer = mnt_layer_name,
                         crs = 2154,
                         res = 25,
                         rgb = FALSE,
                         filename = "mnt.tif",
                         overwrite = TRUE
                         )
  return(mnt)
}

calculate.slope <- function(mnt){
  classes <- c(0, 5, 15, 30, 45, 60, 90)
  pente <- terrain(mnt,
                    v = "slope",
                    unit = "degrees",
                    filename = "pente.tif",
                    overwrite = TRUE
                    )
  pente_classee <- classify(pente, classes)
  return(pente_classee)
}

save.raster.gpkg <- function(SpatRaster) {
  layer_name <- names(SpatRaster)
  writeRaster(SpatRaster,
              filename = "pente.gpkg",
              filetype = "GPKG",
              gdal = c("APPEND_SUBDATASET=YES",
                      paste0("RASTER_TABLE=", layer_name))
  )
}

```

```

save.sf.gpkg <- function(sf) {
  gpkg_path <- paste0(dirname(rstudioapi::getActiveDocumentContext()$path), "/pente.gpkg")
  layer_name <- deparse(substitute(sf))
  st_write(sf,
    gpkg_path,
    layer = layer_name,
    append = TRUE
  )
}

save.gpkg <- function(SpatRaster, sf){
  save.raster.gpkg(SpatRaster)
  save.sf.gpkg(sf)
}

# Fonction principale ----

get.slope <- function(code_post, libelle, section, num_parc){
  code_insee <- code.insee(code_post, libelle)
  zone_parca <- get.cadastre(code_insee, section, num_parc)
  mnt <- get.mnt(zone_parca)
  pente <- calculate.slope(mnt)
  save.gpkg(pente, zone_parca)
  draw(pente, zone_parca)
}

# Exemple ----

code_post = "74420"
libelle <- "SAXEL"
section <- list(NULL)
num_parc <- list(NULL)

get.slope(code_post, libelle, section, num_parc)

```

Calcul des pentes avec QGis et comparaison des deux méthodes

La pente a également été calculée à partir de QGis, avec l'outil calcul de pente. Le MNT utilisé en source est le MNT préalablement téléchargé avec `happign`. Le raster de pentes ainsi obtenu présente des valeurs très proches de celles obtenues avec la fonction `slope()` d'`happign`. La comparaison des deux rasters “pente” a été réalisée avec la calculatrice raster sur QGis. Les variations entre les deux couches sur la zone d’étude considérée sont de l’ordre de 0.02m. Pour la suite et dans l’objectif de l’automatisation de la démarche, nous utiliserons donc le raster pente issu d'`happign`.

Création de la carte d’exploitabilité

Grâce au code R, on a plusieurs couches d’entrée à traiter sur Qgis : La couche du cadastre, la couche des buffers avec les distances aux routes, la couche des pentes, et la couche de la desserte.

Dans un premier temps, on crée une carte du réseau de desserte, une carte des pentes et une carte des distances au sein de la commune. Pour cela, on découpe les couches raster selon une couche de masque, le

cadastral. La symbologie est ensuite ajustée pour créer différentes classes pour les cartes des pentes et des distances.

L'objectif par la suite est de réussir à croiser les données de pente et de distance pour définir une exploitabilité potentielle sur le secteur. Pour cela, un tableau permettant de lier ces deux paramètres est créé, sur la base de celui récupéré sur le site de l'IGN. Il associe une couleur en fonction de la facilité d'exploitation selon le croisement entre la pente et la distance.

Suite à cela, les couches de pentes et de distances sont reclassifiées, afin d'attribuer un numéro aux classes existantes. Par exemple, la couche « pente » est reclassifiées selon le tableau suivant :

De	À	Nouvelle valeur
0	100	1
100	200	2
200	500	3

On a alors 2 nouvelles couches définies selon les nouvelles classes créées. On associe ensuite un numéro aux couleurs choisies dans le tableau, de façon à ce que Vert = 1, Jaune = 2, Orange = 3 et Rouge = 4. On doit ensuite créer une nouvelle couche avec l'outil « calculatrice raster » qui associe à chaque pixel la valeur 1,2,3,4 de la couleur correspondante au croisement dans lequel il se trouve. Etant donné que les conditions sont trop lourdes, on doit diviser cela en plusieurs couches, on a ainsi : - Couche 1 : if("couche_reclassifiee_pente@1" = 1 and "couche_distance_reclassifiee@1" = 1, 1, if("couche_reclassifiee_pente@1" = 1 and "couche_distance_reclassifiee@1" = 2, 2, if("couche_reclassifiee_pente@1" = 2 and "couche_distance_reclassifiee@1" = 3, 3,0)))

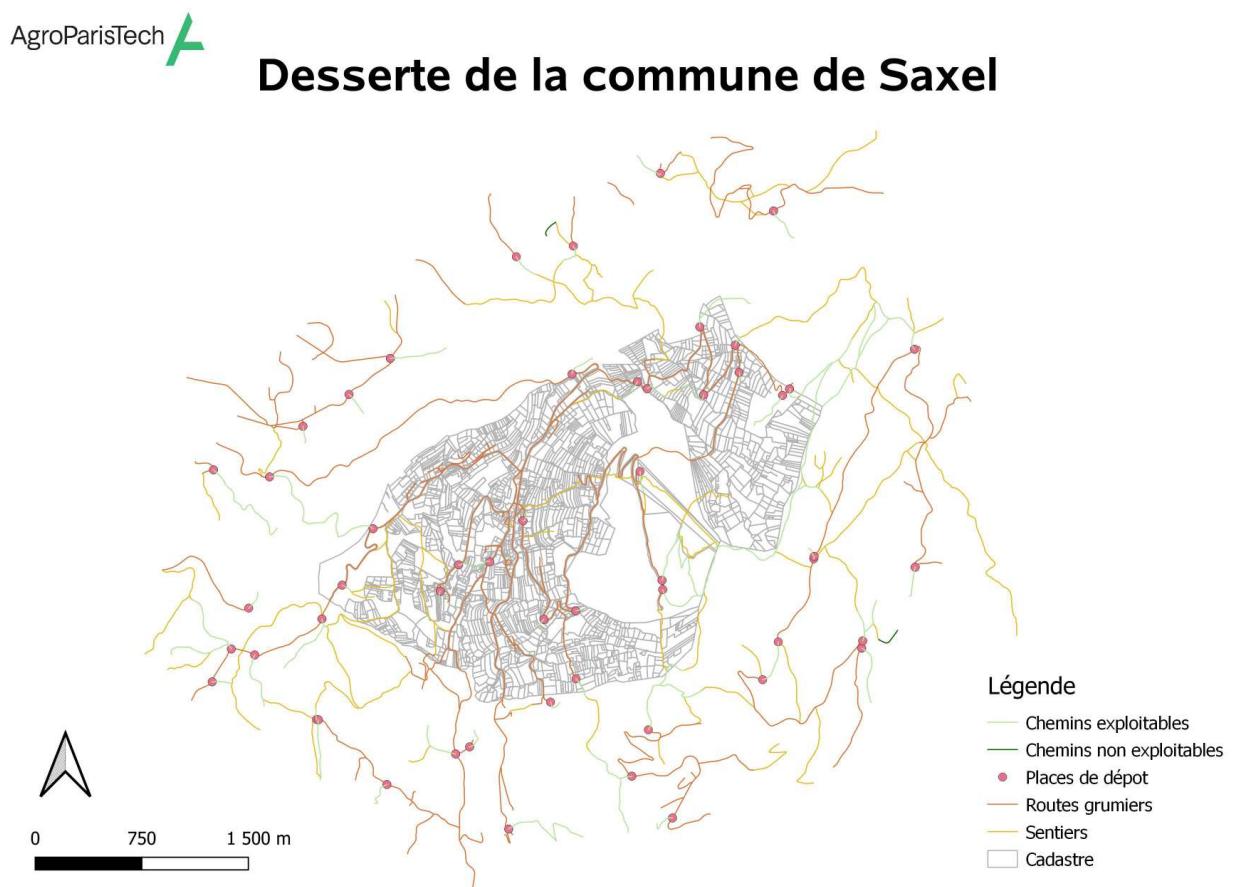
- Couche 2 : if("couche_reclassifiee_pente@1" = 2 and "couche_distance_reclassifiee@1" = 1, 1, if("couche_reclassifiee_pente@1" = 2 and "couche_distance_reclassifiee@1" = 2, 2, if("couche_reclassifiee_pente@1" = 2 and "couche_distance_reclassifiee@1" = 3, 3,0)))
- Couche 3 : if("couche_reclassifiee_pente@1" = 3 and "couche_distance_reclassifiee@1" = 1, 2, if("couche_reclassifiee_pente@1" = 3 and "couche_distance_reclassifiee@1" = 2, 3, if("couche_reclassifiee_pente@1" = 3 and "couche_distance_reclassifiee@1" = 3, 4,0)))
- Couche 4 : if("couche_reclassifiee_pente@1" = 4 and "couche_distance_reclassifiee@1" = 1, 3, if("couche_reclassifiee_pente@1" = 4 and "couche_distance_reclassifiee@1" = 2, 4, if("couche_reclassifiee_pente@1" = 4 and "couche_distance_reclassifiee@1" = 3, 4,0)))
- Couche 5 : if("couche_reclassifiee_pente@1" >= 5, 4, 0) On veut maintenant combiner ces couches ensembles. Pour cela, il faut une couche qui, entre toutes les couches, sélectionne le numéro le plus élevé et attribue cette valeur au pixel concerné. Elle est créé avec « calculatrice raster ». La fonction « MAX » ne fonctionnant pas, la méthode suivante a été utilisée dans la calculatrice : « "couche5_exploit@1" * ("couche5_exploit@1" >= 1) + "couche4_exploit@1" * ("couche5_exploit@1" < 1 and "couche4_exploit@1" >= 1) + "couche3_exploit@1" * ("couche5_exploit@1" < 1 and "couche4_exploit@1" < 1 and "couche3_exploit@1" >= 1) + "couche2_exploit@1" * ("couche5_exploit@1" < 1 and "couche4_exploit@1" < 1 and "couche3_exploit@1" < 1 and "couche2_exploit@1" >= 1) + "couche1_exploit@1" * ("couche5_exploit@1" < 1 and "couche4_exploit@1" < 1 and "couche3_exploit@1" < 1 and "couche1_exploit@1" >= 1) »

La couche finale est ensuite créée, associant donc des valeurs de 1 à 4 pour chacun des pixels. Ces chiffres correspondent aux couleurs du tableau, et représentent l'exploitabilité.

Résultats

Carte de desserte

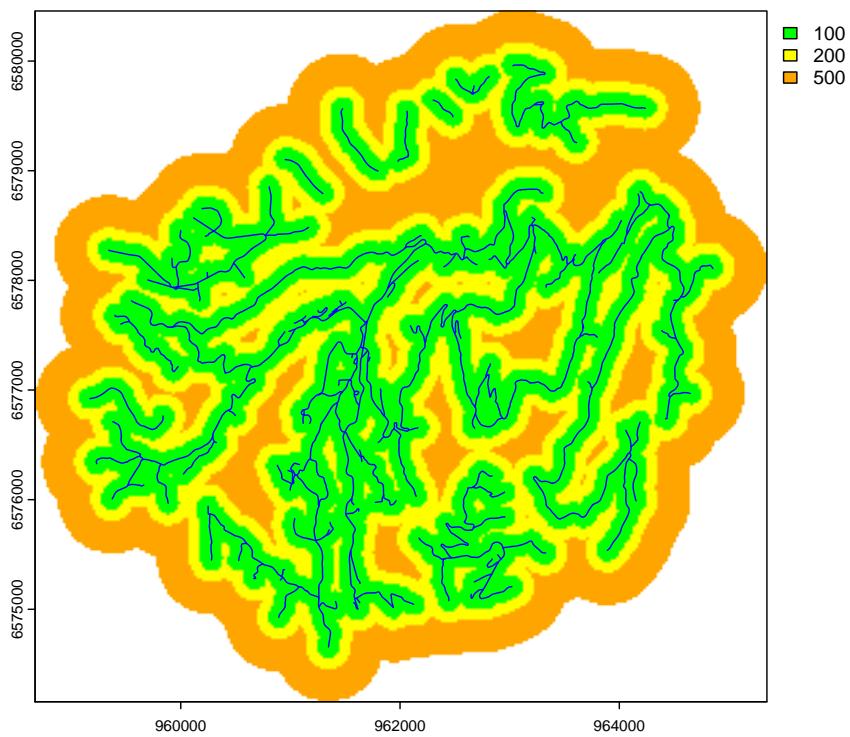
Ci-dessous la carte de la desserte réalisée sur QGis :



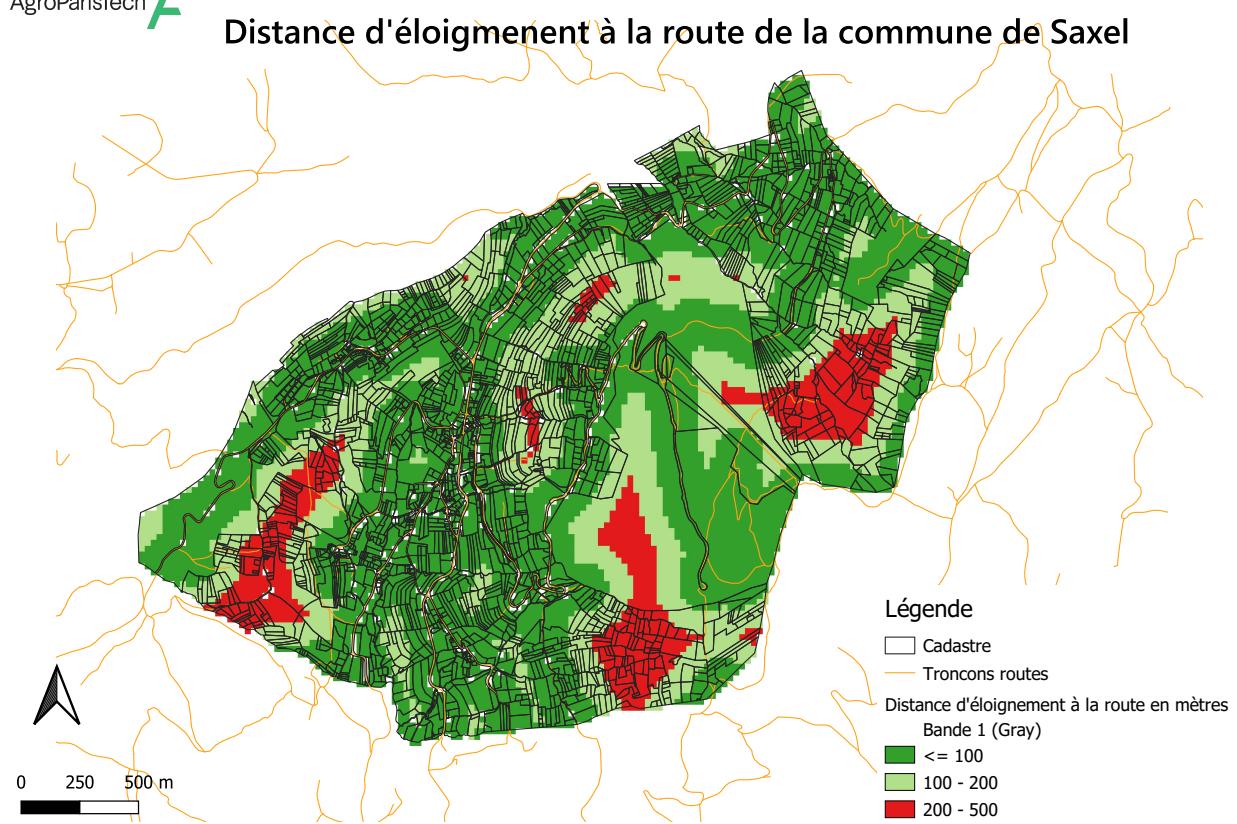
Cartes des buffers autour de la desserte

Ci-dessous la carte des buffers réalisée sur R :

Buffers autour de la desserte



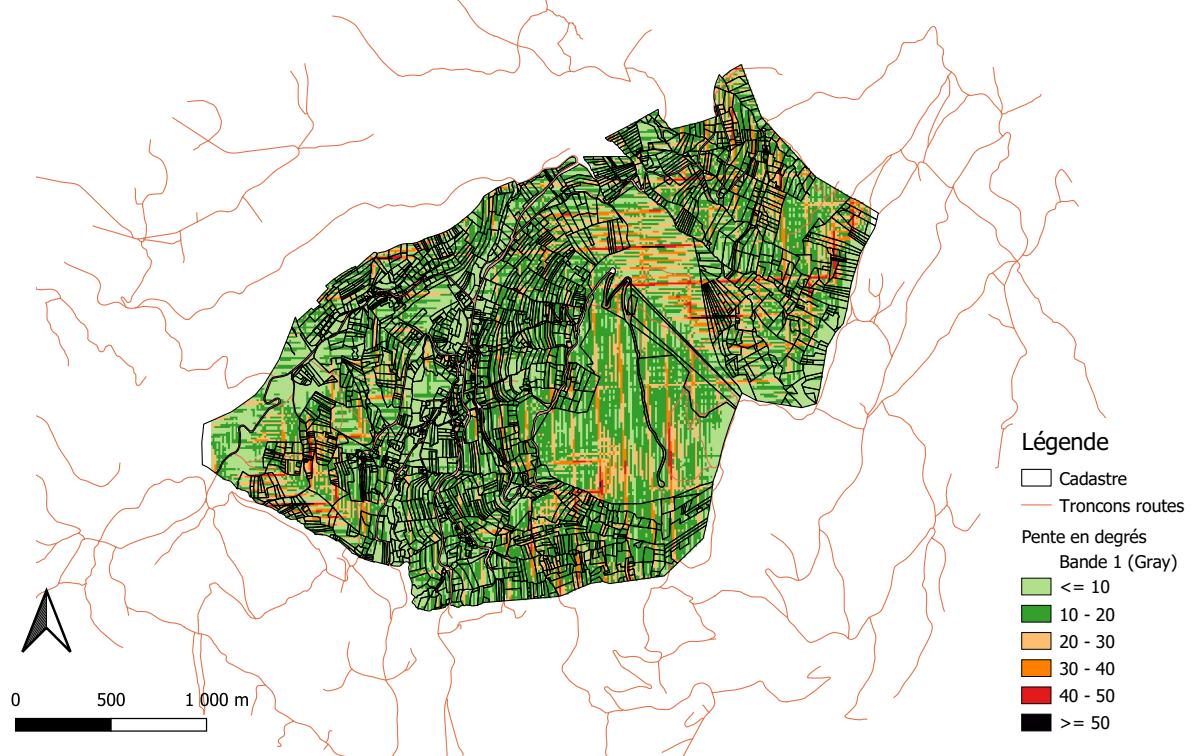
Ci-dessous la carte des buffers réalisée sur QGis :



Carte catégories de pentes

Ci-dessous la carte des catégories de pentes réalisée sur QGis :

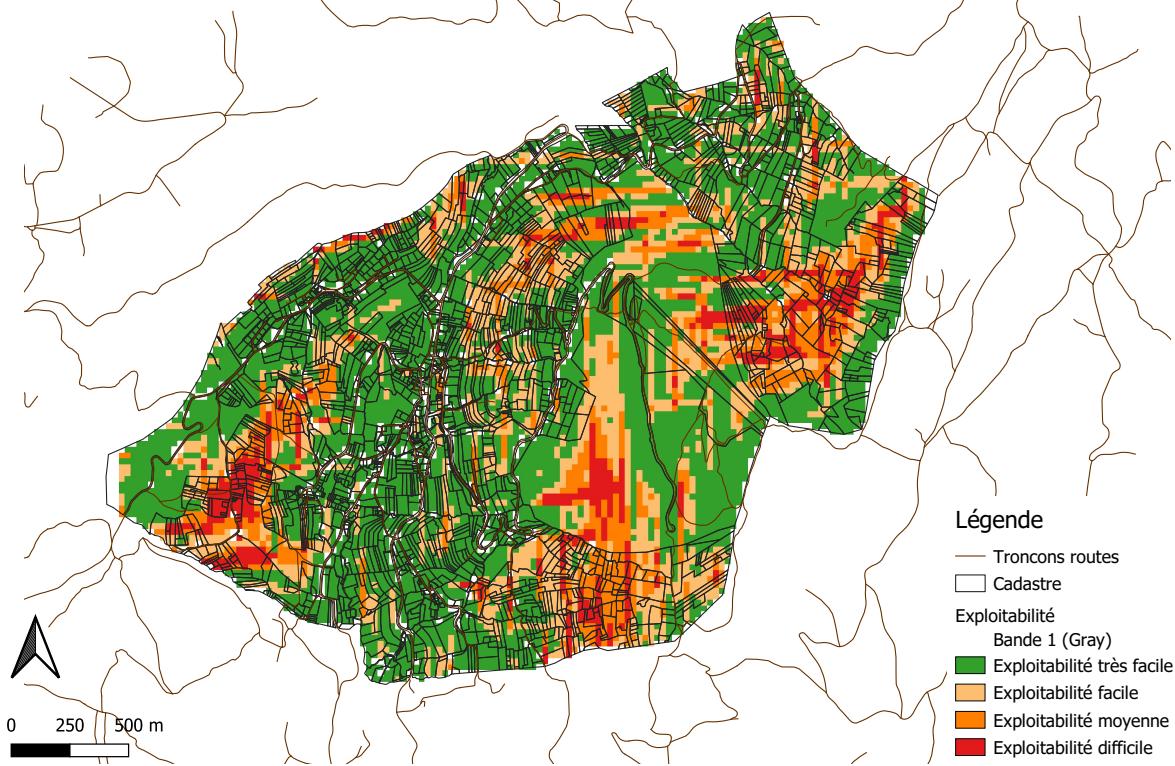
Pentes de la commune de Saxel



Carte d'exploitabilité

Ci-dessous la carte d'exploitabilité réalisée sur QGis :

Exploitabilité des forêts de la commune de Sassel



Les limites et perspectives

L'optimisation des calculs de pente

Bien que deux méthodes de calcul des pentes aient été testées, le MNT utilisé fut toujours le même, à savoir "ELEVATION.ELEVATIONGRIDCOVERAGE" issu de la BD Alti de l'IGN. D'autres sources de MNT pourraient être testées, comme par exemple le RGE ALTI ou le MNT recalculé avec les données LIDAR. De plus, la résolution spatiale utilisée dans l'exemple de ce rapport est de 25m. Il pourrait être envisagé d'intégrer la résolution comme argument de la fonction finale afin de pouvoir l'ajuster selon l'échelle considérée.

Optimisation de la carte d'exploitabilité

Les cartes d'exploitabilité pourraient être améliorées ; la méthode suivante fonctionne car il n'y a que 4 valeurs d'exploitabilité possible, mais s'il y avait eu plus de nuances, cette méthode aurait certainement été trop lourde.

De plus, il faudrait préciser concrètement ce que signifient ces codes couleurs, quelles sont les méthodes d'exploitation derrière, ect.

Les coûts pourraient également être pris en compte, car il existe des parcelles où l'exploitation est possible mais plus onéreuse. Il serait intéressant d'intégrer cela.

Enfin, il faudrait trouver une méthode pour automatiser ce processus dans R, sans avoir à passer par Qgis. Dans l'idéal, tout pourrait être automatisé, dans la mesure où on aurait juste à entrer des informations sur

la parcelles, puis le script récupèrerait les pentes, les distances, et les croiseraient ensembles pour générer une carte d'exploitabilité.

Bien que le temps ait été limité, la méthodologie pour créer une carte d'exploitabilité sur le logiciel R était claire. Étant donné que les données des buffers et des pentes sont sous forme de rasters, avec une résolution de 25 mètres et un système de coordonnées identique, l'analyse croisée de ces rasters est théoriquement faisable. L'objectif était de développer une fonction capable de générer un raster vide aux mêmes dimensions que celui des buffers. Ensuite, pour chaque cellule, une analyse conditionnelle devait être effectuée : par exemple, si le pixel se trouve dans un buffer de 100 mètres et présente une pente inférieure à 30%, la valeur “exploitabilité facile” serait assignnée à ce pixel dans le nouveau raster.

Perspective desserte

L'étape d'acquisition et de caractérisation de la desserte comporte de nombreuses limites et les procédées de traitement de données impliquent de nombreuses hypothèses simplificatrices.

Dans un premier temps, des erreurs ont pu être rencontrées lors de l'import des données IGN. En effet, une première zone délimitée seulement à une dizaine de parcelles de la zone d'étude a fait l'objet de l'import de desserte selon la méthode décrite dans les parties précédentes. Certaines interruptions de routes (tronçons isolés) ont pu être détectées.

Ces erreurs impliquent des corrections manuelles sur Qgis. Or, lors de l'import de la desserte à une plus large échelle (Saxel plus le buffer de 1 km), les erreurs locales de tronçons isolés avaient disparu.

Dans un souci d'universalité du code, notamment en cas de réflexion à des échelles plus fines que celles du massif ou de la commune, il aurait fallu inclure une fonction détectant les tronçons isolés et indiquant à l'opérateur de corriger manuellement les erreurs sur Qgis. Il aurait également fallait prévoir par la suite un input de type shape (couche desserte corrigée).

Un des axes majeurs d'amélioration réside dans la création d'un indicateur attestant de la distance de traîne sur piste jusqu'à une place de dépôt.

Pour cela, deux méthodes sont utilisables :

- La méthode la plus simple (et source de plus d'approximations) consiste en la mise en place de buffers de distance à partir des places de dépôt. Ces classes de distances peuvent notamment être celles tirées du document de l'IGN cité dans les parties précédentes. Ces buffers pourraient être utilisés pour estimer la distance de traîne au niveau de chaque pistes exploitables. Cette distance pourrait ensuite être convertie en durée afin d'obtenir des isochrones. Une amélioration supplémentaire serait d'inclure le facteur coût.

L'objectif serait par la suite de combiner ces buffers (dont l'origine est une place de dépôt) avec le facteur d'exploitabilité à l'échelle de la piste (buffers de distances de débusquage suivant les pistes croisées aux pentes environnantes). Cette opération permettrait une analyse plus précise des contraintes et des capacités opérationnelles des routes et pistes pour les opérations de débardage.

Une autre piste d'amélioration réside dans l'acquisition de données relatives à des infrastructures routières (comme les localisations exactes des places de dépôts et de retournement). Le facteur “sens de circulation” aurait également pu être ajouté (notamment pour les routes à une chaussée et les routes empierrées ne possédant pas de place de retournement).

Une piste d'amélioration supplémentaire aurait été d'indiquer en sortie de la fonction principale modes d'exploitation possibles compte tenu des différentes contraintes topographiques et liées aux infrastructures routières.

L'utilisation des données de desserte provenant d'OpenStreetMap (OSM) présente des limitations notables. Ces données sont souvent incomplètes, difficiles à extraire, et principalement localisées sur les grands axes

routiers et en zones urbaines. Bien qu'OSM puisse être pertinent pour des projets urbains, son utilisation dans les contextes forestiers est inadaptée. En revanche, les données de l'IGN offrent une couverture plus exhaustive et précise, notamment pour les zones rurales et forestières. Elles se superposent souvent aux données OSM, mais surpassent ces dernières en termes de qualité et de fiabilité pour les projets non urbains.



- Carteron, Paul. 2024. “Happign: R Interface to ‘IGN’ Web Services.” [https://CRAN.R-project.org/
package=happign](https://CRAN.R-project.org/package=happign).
- IGN. 2023. “IGD_3.1.1_donnees_exploitabilite.” <https://foret.ign.fr/IGD/fr/indicateurs/3.1.1>.