Simulation et générateurs

Pourquoi simuler ?

La simulation permet de modéliser le comportement d'un système réel complexe.
Difficultés :
- Choix, conception et validation du modèle
- Collecte et analyse des données
- Mise en œuvre informatique longue
- Risque de bugs
- Analyse statistique rigoureuse nécessaire
- Convergence souvent lente

> **i Info**
>
> Simulation gourmande en temps de conception, développement, validation et calcul.

Génération de nombres aléatoires

Un ordinateur étant déterministe, il génère des **pseudo-nombres aléatoires** (PRNG) via des algorithmes à partir d'une **graine** (seed). Même graine → même séquence.

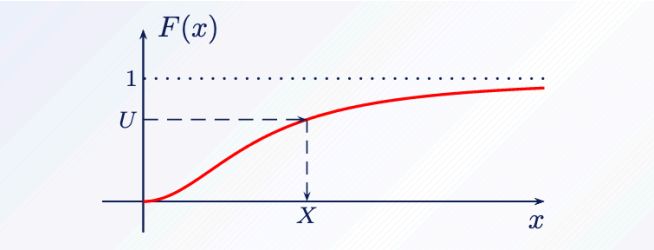
Génération de variables aléatoires

Objectif : représenter des phénomènes aléatoires réels dans une simulation.

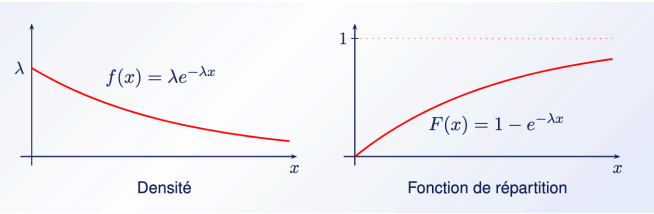
Méthode des fonctions inverses

Utilise la fonction de répartition inverse (quantile) pour générer des variables suivant une distribution donnée.

Variables uniformes : Générer $u \in [0,1]$, puis $x = F^{-1}(u) = a + u(b-a)$



Variables exponentielles : Générer $u \in [0,1]$, puis $x = -\frac{1}{\lambda}\ln(1-u)$



Variables aléatoires discrètes

Méthode des fonctions inverses adaptée : $F^{-1}(u) = \min\{x \mid F(x) \geq u\}$

Utilisation : Générer $u \in [0,1]$, déterminer k tel que $\sum_{i=1}^{k-1} p_i < u \leq \sum_{i=1}^{k} p_i$, retourner $x = x_k$

Variable de Bernoulli $X \sim B(p)$:
- Si $u < p$: succès ($X = 1$)
- Sinon : échec ($X = 0$)

Variable binomiale $X \sim B(n,p)$: Répéter n fois Bernoulli et compter les succès

Variable de Poisson $X \sim P(\lambda)$: Générer des $y_i \sim E(1)$ jusqu'à $\sum_{i=0}^{k} y_i > \lambda$, retourner k

Méthode des mélanges

S'applique quand $F(x) = \sum_{i=1}^{n} p_i F_{i(x)}$ avec $\sum p_i = 1$

Algorithme :
1. Générer indice k selon $P(K = i) = p_i$
2. Générer x selon F_k

Méthode des convolutions

S'applique quand $X = X_1 + X_2 + ... + X_n$

Algorithme : Générer chaque x_i selon X_i, retourner $x = \sum x_i$

Méthode des rejets

Utilise distribution auxiliaire $g(x)$ et constante $M \geq 1$ tel que $f(x) \leq M \cdot g(x)$

Algorithme :
1. Générer z selon $g(x)$
2. Générer $u \sim U(0,1)$
3. Si $u \leq \frac{f(z)}{M \cdot g(z)}$: accepter z
4. Sinon : recommencer

Probabilité d'acceptation = $\frac{1}{M}$

Méthode de Monte-Carlo

Générer un grand nombre de réalisations indépendantes pour estimer des caractéristiques.

Estimateur classique

Espérance : $\hat{\mu} = \overline{x} = \frac{1}{n}\sum_{k=1}^{n} x_k$

Estimateur de la variance

Variance : $\hat{\sigma}^2 = s^2 = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \overline{x})^2$

Précision de l'estimateur

Largeur intervalle de confiance : $\Delta I_c = 2z_{1-\frac{\alpha}{2}}\frac{s}{\sqrt{n}}$

Convergence en $O\left(\frac{1}{\sqrt{n}}\right)$ → diviser par 2 la largeur nécessite 4× plus de réalisations

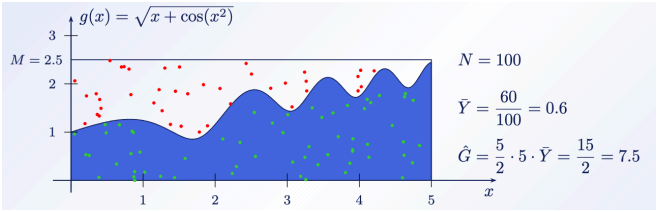
Intervalle de confiance

Avec seuil $1-\alpha$: $I_c = \left[\overline{x} - z_{1-\frac{\alpha}{2}}\frac{s}{\sqrt{n}}; \overline{x} + z_{1-\frac{\alpha}{2}}\frac{s}{\sqrt{n}}\right]$

Seuils courants : 95% ($z = 1.960$), 99% ($z = 2.576$). Nécessite $n \geq 30$ observations.

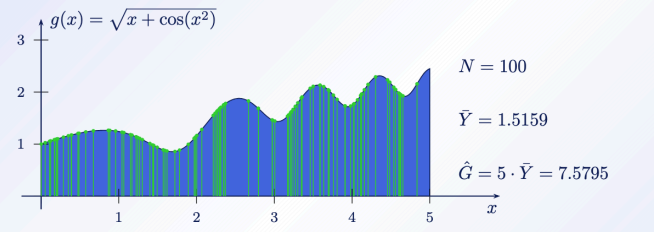
Méthode d'acceptation-rejet

1) Poser $S = 0$ // compte le nombre de succès
2) Pour k de 1 à N faire
3) Générer $X_k \sim \mathcal{U}(a,b)$ et $U_k \sim \mathcal{U}(0,M)$
4) Si $U_k \leq g(X_k)$ poser $Y_k = 1$ // succès (*Hit*)
5) Sinon poser $Y_k = 0$ // échec (*Miss*)
6) Poser $S = S + Y_k$
7) Calculer $\bar{Y} = \frac{S}{N}$ et $\hat{\sigma}_Y^2 = \bar{Y}(1-\bar{Y})$
8) Calculer $\hat{G} = M(b-a)\bar{Y}$ // estimateur de $G = \int_a^b g(x)dx$
9) et $\hat{\sigma}_{\hat{G}} = M(b-a)\sqrt{\frac{\hat{\sigma}_Y^2}{N}}$ // estimateur de l'écart-type de $\hat{G}$
10) Retourner $\hat{G}$ et l'intervalle de confiance au seuil $1-\alpha$: $\left[\hat{G} \pm z_{1-\frac{\alpha}{2}}\hat{\sigma}_{\hat{G}}\right]$



Échantillonnage uniforme

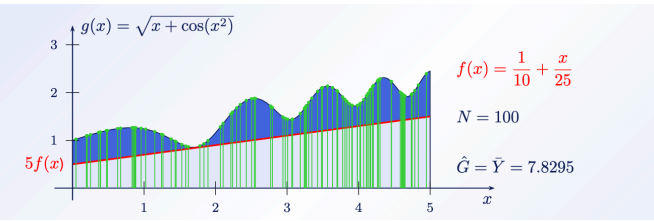
1) Poser $S = 0$ et $Q = 0$ // sommes des Y_k et de leurs carrés
2) Pour k de 1 à N faire
3) Générer $X_k \sim \mathcal{U}(a,b)$
4) Calculer $Y_k = g(X_k)$
5) Poser $S = S + Y_k$ et $Q = Q + Y_k^2$
6) Calculer $\bar{Y} = \frac{S}{N}$ et $\hat{\sigma}_Y^2 = \frac{Q}{N} - \bar{Y}^2$
7) Calculer $\hat{G} = (b-a)\bar{Y}$ // estimateur de $G = \int_a^b g(x)dx$
8) et $\hat{\sigma}_{\hat{G}} = (b-a)\sqrt{\frac{\hat{\sigma}_Y^2}{N}}$ // estimateur de l'écart-type de $\hat{G}$
9) Retourner $\hat{G}$ et l'intervalle de confiance au seuil $1-\alpha$: $\left[\hat{G} \pm z_{1-\frac{\alpha}{2}}\hat{\sigma}_{\hat{G}}\right]$



Échantillonnage préférentiel

Génère des points dans les régions favorisant les points contribuant le plus à la valeur de G.

1) Poser $S = 0$ et $Q = 0$ // sommes des Y_k et de leurs carrés
2) Pour k de 1 à N faire
3) Générer X_k selon la densité $f(x)$
4) Calculer $Y_k = \dfrac{g(X_k)}{f(X_k)}$
5) Poser $S = S + Y_k$ et $Q = Q + Y_k^2$
6) Calculer $\bar{Y} = \dfrac{S}{N}$ et $\hat{\sigma}_Y^2 = \dfrac{Q}{N} - \bar{Y}^2$
7) Calculer $\hat{G} = \bar{Y}$ // estimateur de $G = \int_a^b g(x)dx$
8) et $\hat{\sigma}_{\hat{G}} = \sqrt{\dfrac{\hat{\sigma}_Y^2}{N}}$ // estimateur de l'écart-type de $\hat{G}$
9) Retourner $\hat{G}$ et l'intervalle de confiance au seuil $1 - \alpha$: $\left[\hat{G} \pm z_{1-\frac{\alpha}{2}} \hat{\sigma}_{\hat{G}}\right]$



Optimisation combinatoire

Caractérisée par un ensemble de solutions finies mais très grand :
- Ensemble **fini** S de solutions admissibles
- Fonction objectif $f : S \to \mathbb{R}$

Objectif : $\min_{x \in S} f(x)$

Types de problèmes

Faciles (polynomiaux) : solution optimale en temps polynomial
- Arbre recouvrant de poids minimal
- Plus court chemin (Dijkstra, Bellman-Ford)

Difficiles (NP-difficiles) : pas d'algorithme polynomial connu
- Voyageur de commerce (TSP)
- Stable de cardinal maximum

Algorithmes de résolution

Algorithmes exacts : fournissent toujours une solution optimale
- Problèmes faciles : complexité polynomiale
- Problèmes difficiles : complexité exponentielle

Algorithmes approximatifs : solution réalisable mais pas forcément optimale
- Assurent qualité minimale
- Complexité raisonnable (polynomiale)

Algorithmes heuristiques : solution sous-optimale en général
- Aucune garantie sur la qualité
- Efficaces en pratique
- Fournissent souvent de bonnes solutions empiriquement

Types d'heuristiques

Heuristiques constructives : construisent une solution pas à pas en ajoutant des éléments

Heuristiques d'amélioration : partent d'une solution admissible et cherchent à l'améliorer en explorant son voisinage

Heuristiques de coloration

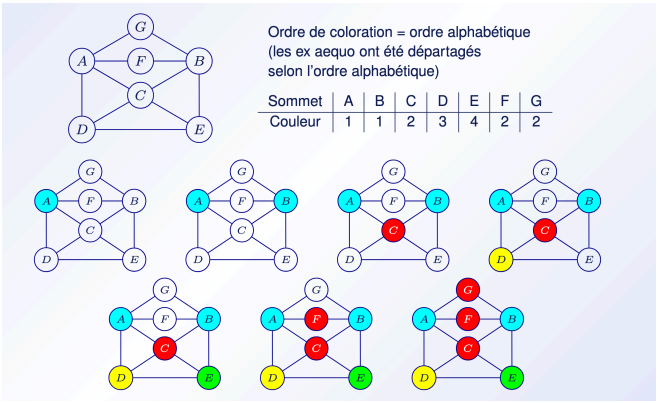
Coloration d'un graphe

Soit $G = (V, E)$ un graphe non orienté. Une **coloration** affecte des couleurs aux sommets tels que deux sommets adjacents n'ont pas la même couleur. But : minimiser le nombre de couleurs.

Heuristique LF (largest-first)

Ordonner les sommets par ordre décroissant de degré et les colorier dans cet ordre.

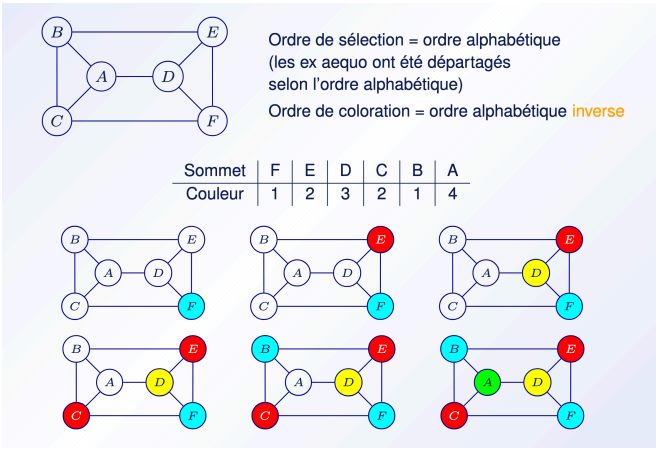
Étapes :
1. Calculer le degré de chaque sommet
2. Trier les sommets par degré décroissant
3. Pour chaque sommet dans l'ordre :
   - Assigner la plus petite couleur non utilisée par ses voisins
   - Si toutes les couleurs sont utilisées, créer une nouvelle couleur



Heuristique SL (smallest-last)

Colorier le plus petit sommet (degré minimal) dans le sous-graphe restant, puis le retirer. Répéter jusqu'à tous les sommets colorés. Coloration dans l'ordre inverse de suppression.

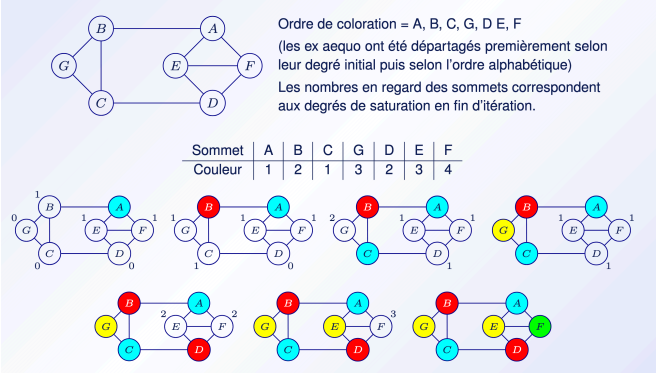
Étapes :
1. Initialiser une pile vide
2. Tant qu'il reste des sommets :
   - Trouver le sommet de degré minimal dans le sous-graphe restant
   - Empiler ce sommet
   - Retirer le sommet du graphe (et ses arêtes)
3. Tant que la pile n'est pas vide :
   - Dépiler un sommet
   - Lui assigner la plus petite couleur non utilisée par ses voisins

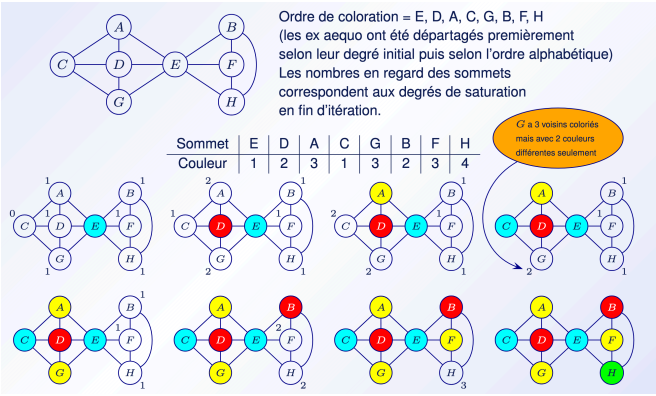


Heuristique DSATUR

Colorier les sommets selon leur degré de saturation : nombre de couleurs différentes déjà utilisées par leurs voisins.

Étapes :
1. Colorier le sommet de degré maximal avec la couleur 1
2. Pour chaque sommet non coloré, calculer DSAT = nombre de couleurs distinctes utilisées par ses voisins
3. Choisir le sommet avec DSAT maximal (en cas d'égalité, choisir celui de degré maximal)
4. Lui assigner la plus petite couleur non utilisée par ses voisins
5. Répéter les étapes 2-4 jusqu'à ce que tous les sommets soient colorés





Heuristique RLF (recursive largest-first)

Construit séquentiellement une partition du graphe en sous-ensembles stables $\{C_1, ..., C_k\}$, chaque ensemble C_i recevant la couleur i.

Principe : Pour chaque couleur, construire le plus grand ensemble stable possible en sélectionnant les sommets qui ont le plus de voisins déjà exclus.

Étapes détaillées :

1. **Initialisation** : Choisir le sommet v de degré maximal (le plus connecté)
2. **Créer les ensembles** :
   - $C := \{v\}$: ensemble des sommets qui recevront la couleur actuelle
   - $U := \text{Adj}[v]$: sommets adjacents à C (exclus pour cette couleur)
   - $W := V \setminus (\text{Adj}[v] \cup \{v\})$: sommets candidats (ni dans C, ni adjacents à C)
3. **Agrandir l'ensemble stable C** :
   - Tant que W non vide :
     - Choisir $w \in W$ ayant le **maximum** de voisins dans U
     - Ajouter w à C
     - Mettre à jour $U := U \cup \text{Adj}[w]$ (ajouter les voisins de w)
     - Mettre à jour $W := W \setminus (\text{Adj}[w] \cup \{w\})$ (retirer w et ses voisins)
4. **Assigner la couleur** : Tous les sommets de C reçoivent la même couleur
5. **Récursion** : Répéter avec les sommets restants jusqu'à ce que tous soient colorés

Remarques performances

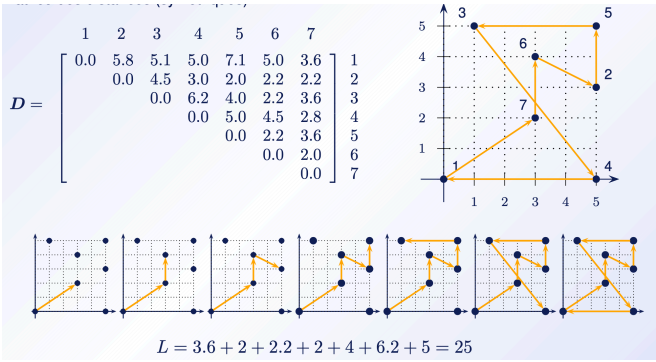
1. Les 4 heuristiques utilisent au plus $\Delta(G) + 1$ couleurs
2. Performance vérifiée par tous les algorithmes gloutons
3. DSATUR et RLF optimales pour graphes bipartis (≤ 2 couleurs)
4. DSATUR et RLF empiriquement supérieures mais plus lentes

Heuristiques pour le TSP

Problème du voyageur de commerce : trouver la tournée la plus courte visitant chaque ville une seule fois.

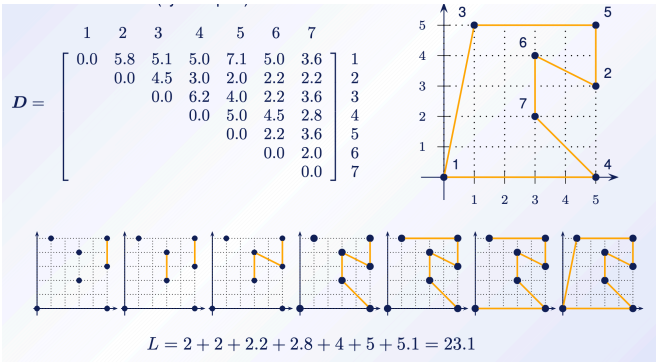
Plus proche voisin

1. Choisir ville de départ aléatoirement
2. Visiter ville la plus proche non visitée
3. Répéter jusqu'à toutes visitées
4. Retourner à la ville de départ



Heuristique gloutonne

1. Ensemble vide d'arêtes
2. Trier arêtes par coût croissant
3. Ajouter arêtes sans créer cycle prématuré ni degré > 2
4. Répéter jusqu'à toutes villes connectées



Insertion la moins coûteuse

1. Tournée partielle : ville + plus proche voisine
2. Pour chaque ville non incluse : calculer coût d'insertion
3. Insérer ville avec augmentation de coût minimale
4. Répéter jusqu'à toutes incluses

Insertion la plus coûteuse

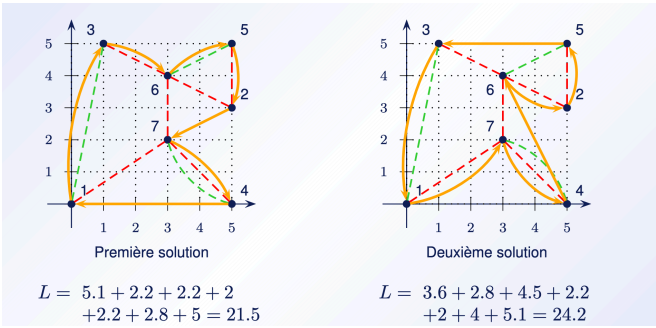
Même principe que la moins coûteuse, mais insérer la ville avec augmentation de coût maximale.

Christofides

Garantit solution $\leq 1{,}5\times$ coût optimal (avec inégalité triangulaire).

Principe : Transformer cycle eulérien en cycle hamiltonien.

1. Arbre recouvrant T de poids minimum
2. Couplage parfait M de poids minimum sur sommets degré impair de T
3. Ajouter arêtes de M à $T \rightarrow$ graphe avec tous sommets degré pair $\rightarrow$ cycle eulérien
4. Parcourir cycle avec raccourcis pour éviter visites multiples



Meilleurs fusions

1. Chaque ville = sous-tour individuel
2. Calculer coût de fusion pour chaque paire
3. Fusionner paire avec plus faible augmentation
4. Répéter jusqu'à un seul tour