

# Projets :

## 1 Présentation générale des projets

Ces projets visent à présenter différentes techniques algorithmiques au grand public (on visera dans un premier temps un public averti, type lycéen) et quels sont les “challenges” dans ce domaine (e.g., un algorithme/ordinateur ne peut pas tout faire!!! ... même les LLM<sup>1</sup>!!!).

Chaque projet traitera d’un problème particulier (des problèmes d’optimisation et/ou d’existence) pour lequel il s’agira de développer une application ludique et esthétique qui permette à l’utilisateur d’essayer de résoudre (et donc de comprendre) le problème ainsi qu’un (ou des) algorithme(s) pour résoudre le problème. Certains des problèmes proposés sont des jeux à deux joueurs auquel cas, les utilisateurs devront pouvoir jouer l’un contre l’autre ou contre la machine.

Chaque sujet est pensé pour décrire une technique algorithmique spécifique. L’autre aspect (complémentaire) du projet sera de concevoir une activité débranchée permettant d’expliquer la technique algorithmique sous-jacente.

Un autre aspect important qui doit être considéré (et donc qui devra être expliqué de façon “vulgarisée”) est la notion de complexité algorithmique et de difficulté (polynomial, NP-difficile,...) des problèmes.

Il est bien sûr attendu que les étudiants effectuent une recherche bibliographique sur le problème considéré et les différentes techniques pour le résoudre.

### 1.1 Rendus attendus

- Une application illustrant le problème. Cette application doit être :
  - interactive : permettant à un/des utilisateur(s) de proposer des instances, de tester des solutions ;
  - avec des méthodes de résolution automatiques (exactes ou approchées). Au moins une méthode efficace (i.e., rapide) est attendue ;
  - une façon de comparer les résultats (temps d’exécution, qualité de la solution...).

Une jolie interface visuelle pour l’application sera un plus mais n’est pas imposée et il est suggéré de commencer par une version en lignes de commande.

Dans tous les cas, commencer par réfléchir à la représentation du problème, d’une solution au problème (structures de données) et à une méthode pour tester/évaluer une solution.

- Le prototype d’une activité débranchée pour “vulgariser” votre projet ;
- Un rapport final (max 10 pages + annexes possibles), et
- Une présentation (e.g. powerpoint) pour la dernière séance (entre 20 et 30 minutes par groupe, questions comprises, selon le nombre de groupes).

---

1. Je suis curieux de savoir combien d’entre vous pensent que “l’IA” peut tout faire ?

## 1.2 À propos des applications attendues / contraintes techniques

Les applications attendues ont vocation à être déployées sur le serveur Terra Numerica pour être rendues publiques aux côtés des ateliers existants. Il faudra donc tenir compte des contraintes liées aux capacités mémoire et CPU du système. La technologie utilisée pour les stacks (i.e. appli avec frontend et backend) est Docker Compose. Les stacks doivent être autoportants, c'est à dire ne pas dépendre d'un serveur (base de données ou autre) extérieur. Ils devront donc inclure ce type de ressource sous forme de containers. Les applications de type site statique sont pour leur part déployées dans un sous-répertoire dédié à l'application au sein de l'espace de stockage d'un serveur Nginx, aux côtés d'autres applications de même nature. Elles seront donc livrées sous forme d'une archive qui y sera décompactée. Il faudra tenir compte des impacts de ce type de déploiement sur la structure des path des URLs.

## 1.3 Présentation générale des projets

Dans ce qui suit, nous présentons les 8 projets (l'ordre dans lequel ils sont présentés est arbitraire). Ils sont présentés de façon mathématique pour être à la fois succinct et précis. Une telle présentation est bien sûr destinée à des étudiants universitaires, mais pas au grand public. Un de vos objectifs sera de trouver une façon "simple" et ludique de les présenter au grand public. Dans certains cas, nous donnons des exemples de présentation au grand public mais laissez libre votre imagination pour en inventer d'autres. Dans tous les cas, nous présentons des exemples d'applications des problèmes considérés et donnons des références pour vous aider à débiter votre projet. Dans certains cas, nous suggérons également certains algorithmes qui doivent être implémentés : ces algorithmes doivent être compris, i.e., vous devez être capables de les expliquer (pourquoi ils sont corrects ? quelles sont leurs performances ?...) à vos (grands) parents, frères et sœurs, ami(e)s... !

Ne vous contentez pas des références ci-dessous, faites vos propres recherches bibliographiques ! Outre vos compétences "techniques", vous serez évalués sur la curiosité dont vous aurez fait preuve vis-à-vis de votre sujet.

**Conseil :** Une dernière remarque avant de commencer : communiquez entre groupes !! renseignez vous sur le sujet des autres, discutez entre vous !! Certains projets ont des "briques de bases" communes...

**Dernières remarques triviales sur votre démarche et sur la notation :** Préférez toujours nous poser des questions (de vive voix ou par email) ou faire des recherches sur Internet (nous vous donnons plusieurs références dans ce but) plutôt que demander à votre LLM préféré. Si (quand) vous demanderez à un LLM, vérifiez et comprenez la solution proposée. Si vous nous présentez un projet génial mais que vous n'êtes pas capables de nous l'expliquer, vous aurez au plus 5/20... Enfin, **les notes seront individuelles**, pas par groupe... Travailler en groupe signifie se partager les tâches, pas qu'une ou deux personnes travaillent pendant que les autres se tournent les pouces...

**Glossaire :** Dans ce qui suit, lorsque l'on dit "dans un premier temps" ou "on commencera par...", cela signifie généralement que c'est le B.A.BA et que l'on espère que cela soit fait dans les premières semaines (premier tiers ou première moitié) du projet.

## 2 Ordonnancement, Cuisine et approximations

Commençons par le problème d'équilibrage de charge (**load balancing**). Supposons qu'on dispose d'un ensemble de  $n$  fruits (ex : 43 pommes, 57 mangues, 64 poires, 107 tomates, 13 litchis...) et d'un ensemble de  $m$  commis de cuisine. Chaque type de fruit demande un temps (fixe) pour être épluché et coupé. Par exemple, disons qu'il faut  $t_{pomme} = 30$  secondes pour éplucher et couper une pomme et  $t_{mangue} = 10$  minutes pour s'occuper d'une mangue<sup>2</sup>. Un commis ne peut s'occuper que d'un fruit à la fois et lorsqu'il commence un fruit, il le traite jusqu'au bout. La question est de répartir les fruits entre les commis de sorte que tous les fruits soient traités en le moins de temps total possible. C'est-à-dire qu'on veut diviser l'ensemble des fruits en différentes parties et donner chaque partie à un commis (qui va s'occuper de sa partie, i.e. de son ensemble de fruits) et on veut que l'ensemble des fruits soit épluché/coupé le plus vite possible.

Mathématiquement :

**Entrées :**  $(t_1, \dots, t_n) \in (\mathbb{R}^+)^n$  et  $m \in \mathbb{N}$ .

**Sortie :** Une partition  $(A_1, \dots, A_m)$  de  $\{1, \dots, n\}$  telle que  $\max_{i \leq m} \sum_{j \in A_i} t_j$  est minimum.

Le problème d'équilibrage de charge est NP-complet et de simples algorithmes gloutons donnent de bonnes approximations. Voir Chapitre 3 de <https://www-sop.inria.fr/members/Nicolas.Nisse/LectureGraphAlgo.pdf>.

Voir aussi (même si, à première vue, un peu différent) [https://www.youtube.com/watch?v=Vo4Kb0Vw\\_HI](https://www.youtube.com/watch?v=Vo4Kb0Vw_HI).

Complexifions un peu en introduisant des relations (**ordonnancement**) entre les tâches à effectuer (jusque là, l'ordre dans lequel les tâches étaient effectuées n'était pas important : un commis pouvait éplucher une pomme avant une mangue ou vice-versa sans impacter le temps de préparation).

On dispose maintenant d'un ensemble de plat (chaque plat a un temps de préparation et un temps de cuisson, il doit être préparé avant d'être cuit), d'un ensemble de  $m$  commis et d'un ensemble de  $x$  fours. Comment répartir les différents plats aux commis et aux fours de façon à ce que tous les plats soient prêts dans les plus brefs délais ?

Donnons 2 exemples (tirés de l'exposition <https://www.fondation-blaise-pascal.org/success-story-exposition-dans-ma-cuisine/>) où nous supposons qu'il n'y a que  $m = 1$  commis et que  $x = 1$  four.

**Exemple 1 :** 3 plats : le premier plat a un temps de préparation de 15 minutes et un temps de cuisson de 17 minutes. Le second plat a un temps de préparation de 11 minutes et un temps de cuisson de 16 minutes. Le dernier plat a un temps de préparation de 0 minutes et un temps de cuisson de 12 minutes. Saurez vous trouver une solution optimale ?

**Exemple 2 :** 3 plats : le premier plat a un temps de préparation de 16 minutes et un temps de cuisson de 10 minutes. Le second plat a un temps de préparation de 14 minutes et un temps de cuisson de 14 minutes. Le dernier plat a un temps de préparation de 11 minutes et un temps de cuisson de 8 minutes. Saurez vous trouver une solution optimale ?

**Exemple 3 :** 6 plats : le premier plat a un temps de préparation de 8 minutes et un temps de cuisson de 12 minutes. Le plat 2 a un temps de préparation de 12 minutes et un temps de cuisson de 8 minutes. Le plat 3 a un temps de préparation de 17 minutes et un temps de cuisson de 20 minutes. Le plat 4 a un temps de préparation de 19 minutes et un temps de

---

2. vous savez faire plus vite??

cuisson de 17 minutes. Le plat 5 a un temps de préparation de 19 minutes et un temps de cuisson de 20 minutes. Le plat 6 a un temps de préparation de 19 minutes et un temps de cuisson de 12 minutes. Saurez vous trouver une solution optimale ?

Le dernier exemple semble beaucoup plus difficile à résoudre “à la main”. Un plus pour ce projet serait de proposer une méthode pour générer ce type d'exemples “difficiles”.

**Autre approche ludique :** Supposons que vous planifiez vos vacances : vous avez une semaine de 5 jours (lundi à vendredi) avec 7 heures ouvrables par jour (disons entre 9h et 16h). Vous disposez d'un ensemble d'activités (parc d'attraction, musée, magasin, plage...) que vous pouvez choisir (ou pas) et dont vous devez décider quand les faire (si vous les avez choisies). Chaque activité a des horaires d'ouverture (e.g., : le musée est ouvert de 10h à 15h sauf le lundi où il est fermé), un prix (e.g., la plage coûte 10 euros (la glace que vous y achetez), le musée 15 euros et le parc d'attraction 70 euros) et un taux de satisfaction (e.g., le parc d'attraction vous plait 5 fois plus que faire du shopping). Supposez que vous avez un budget fixe, comment optimiser vos vacances ?

**Notions abordées :** Problèmes d'optimisation, ordonnancement, algorithme glouton, approximation, NP-difficile.



FIGURE 1 – Photos de l'exposition “Dans ma Cuisine” de la MMI.

### 3 Sudoku and beyond

Une fois n'est pas coutume, nous commençons par définir le problème de la façon la plus générale (?) possible (et donc, il vous faudra attendre les exemples suivants pour que cela soit plus clair...).

On considère une grille  $n \times m$  (par exemple, une grille classique de Sudoku est  $9 \times 9$ ; un échiquier est une grille  $8 \times 8$ ...). La case  $(i, j)$  est celle à la ligne  $i$  et à la colonne  $j$ . On dispose également, d'un ensemble de symboles (e.g., pour le sudoku  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ , pour les échecs  $\{pion, roi, reine, tour, fou...\}$ ). L'objectif est d'attribuer à chaque case  $c = (i, j)$  un unique symbole  $symb(c)$  de sorte que les toutes contraintes soient satisfaites. Il y a deux types de contraintes :

**Contrainte locale :** Étant donnée une case  $c = (i, j)$ , soit  $\mathcal{L}_c$  un ensemble de cases. Une contrainte locale est du type : pour toute case  $c' \in \mathcal{L}_c$ ,  $symb(c')$  et  $symb(c)$  satisfont une certaine relation (par ex,  $symb(c') \neq symb(c)$  ou, (si  $symb(c)$  est pair alors  $symb(c')$  est pair)...);

**Contrainte globale :** Soit  $\mathcal{L}$  un ensemble de cases. Une contrainte globale est du type : l'ensemble  $\{symb(c) \mid c \in \mathcal{L}\}$  satisfait une certaine relation. Par exemple,  $|\mathcal{L}| = |\{symb(c) \mid c \in \mathcal{L}\}|$  (donc les symboles attribués aux cases de  $\mathcal{L}$  sont deux-à-deux distincts...).

**Problème :** Étant donné une grille  $n \times m$ , un ensemble de symboles et un ensemble de contraintes (règles) locales et globales, et un certain nombre de cases (possiblement aucune) auxquelles on a attribué des symboles, comment trouver une affectation valide de symboles aux cases restantes ?

La présentation ci-dessus est volontairement très générale (et donc, malheureusement un peu difficile à suivre) pour vous aider à proposer un algorithme de résolution le plus général possible (qui puisse être adapté facilement selon les contraintes). Ci-dessous, nous donnons des exemples concrets qui, nous l'espérons, permettront une meilleure compréhension du problème.

#### 3.1 Tango

Le jeu Tango considère une grille  $6 \times 6$ . Chaque cellule peut être un soleil ou un lune. Les contraintes globales sont que :

- chaque ligne doit contenir exactement 3 lunes et 3 soleils ;
- chaque colonne doit contenir exactement 3 lunes et 3 soleils ;
- chaque ensemble de 3 cellules consécutives horizontalement contient au moins une lune et au moins un soleil ;
- chaque ensemble de 3 cellules consécutives verticalement contient au moins une lune et au moins un soleil.

Les 2 dernières contraintes disent simplement qu'il ne peut y avoir 3 symboles identiques alignés ni horizontalement ni verticalement.

Dans certaines instances, des contraintes locales sont ajoutées : 2 cellules adjacentes doivent recevoir un même symbole (représenté par "=") ou des symboles différents (représenté par "×").

Toutes les instances proposées en ligne (eg. Tango) semblent pouvoir être résolues par un algorithme glouton (vous devrez en concevoir un). Est-ce toujours le cas ? et sinon, comment générer des instances qui puissent assurément être résolues de façon gloutonne ?

## 3.2 Sudoku

Le jeu Sudoku est un jeu très connu qui considère une grille  $9 \times 9$  partitionnée en 9 sous-grilles  $3 \times 3$ . Les symboles sont les entiers de 1 à 9. Les contraintes (uniquement globales) sont que :

- chaque ligne doit contenir exactement une fois chaque symbole ;
- chaque colonne doit contenir exactement une fois chaque symbole ;
- chaque sous-grille doit contenir exactement une fois chaque symbole.

Proposer un/des algorithm(e)s pour résoudre une grille de Sudoku (vous pouvez généraliser à de plus grandes grilles). Comment générer des instances qui puissent assurément être résolues de façon gloutonne ?

## 3.3 Et au delà

Ici, nous vous invitons à voir ce site où de nombreuses généralisations intéressantes et amusantes sont présentées comme : <https://www.youtube.com/watch?v=ZxElsPvjKqW> ou <https://www.youtube.com/watch?v=u0FhERdlWfc> ou [https://www.youtube.com/watch?v=6o7caUPFY\\_s....](https://www.youtube.com/watch?v=6o7caUPFY_s....)

Dans ces jeux, en plus des règles classiques du Sudoku, on ajoute des contraintes globales (par exemple, en définissant des zones qui ne peuvent pas contenir 2 fois un même symbole) et locales (par exemple la règle du cavalier : pour chaque case  $c$ , toute case  $c'$  accessible depuis  $c$  par un mouvement de cavalier (aux échecs) doit recevoir un symbole différent de celui de  $c$ ).

### Aller plus loin ?

- Sauriez vous “vulgariser” le problème du millénaire  $P = NP$ ? en utilisant ce type de problèmes ? Expliquer pourquoi étudier ces “jeux” a un impact important sur les techniques de chiffrement (bancaire, des emails...)?
- Ce type de problème peut être (efficacement ou pas) résolu par des méthodes de programmation linéaire (LP, Dantzig) ou de programmation par contraintes (CP). Sauriez vous “vulgariser” ces méthodes ? Quelle est leur importance pratique (e.g., lien entre programmation linéaire et planification lors de la seconde guerre mondiale)...

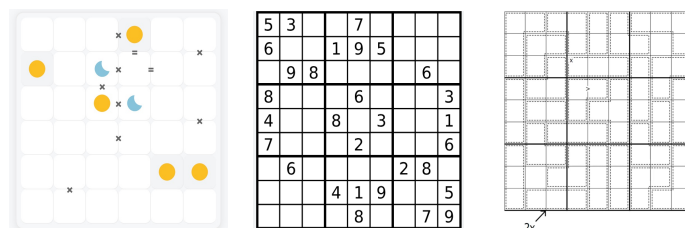


FIGURE 2 – Exemples. Tango (gauche) ; Sudoku (milieu) ; et au delà (droite)...

**Notions abordées :** Algorithme glouton, recherche exhaustive, NP-difficile, LP/ILP, CP.

## 4 Voyageur de commerce et algorithmes génétiques

On considère un ensemble de points dans le plan, qui correspondent à des villes. Un des points abrite la résidence d'un VRP qui doit déterminer son trajet de la journée : en partant de sa résidence, il doit visiter chacune des autres villes pour finalement rentrer chez lui le soir. Le but est de calculer un tel trajet qui soit le plus court possible.

On peut représenter une instance par un graphe  $G = (V, E)$  complet avec  $n$  sommets  $V = \{v_1, \dots, v_n\}$ , chaque sommet représentant un point (une ville). La résidence du VRP est  $v_1$ . Pour chaque paire de sommets  $v_i$  et  $v_j$ , l'arête  $\{v_i, v_j\} \in E$  est étiquetée par la distance euclidienne  $dist(v_i, v_j)$  entre les points correspondants. Mathématiquement, le problème est alors :

**Entrées :**  $G = (V, E)$ ,  $V = \{v_1, \dots, v_n\}$ ,  $dist : E \rightarrow \mathbb{R}^+$  vérifiant l'inégalité triangulaire.

**Sortie :** une permutation  $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$  avec  $\pi(1) = 1$  et telle que

$$\left( \sum_{1 \leq i < n} dist(v_{\pi(i)}, v_{\pi(i+1)}) \right) + dist(v_{\pi(n)}, v_{\pi(1)}) \text{ est minimum.}$$

Il s'agit d'un problème classique NP-complet pour lequel il existe des algorithmes exact par programmation dynamique en temps  $O(2^n)$ , une 2-approximation simple basée sur l'algorithme de Kruskal et une  $\frac{3}{2}$ -approximation (algorithme de Cristofides). Voir Chapitre 4 de <https://www-sop.inria.fr/members/Nicolas.Nisse/LectureGraphAlgo.pdf>.

On pourra commencer par proposer une interface qui permette aux utilisateurs de placer des points et de proposer une solution (dont la longueur sera évaluée). Ensuite, vous implémenterez les algorithmes classiques de façon à obtenir des solutions de référence.

Cependant, l'objectif principal de cette activité est de présenter la notion d'algorithmes génétique. Plus généralement, on veut montrer comment "attaquer" un problème NP-complet lorsqu'on ne désire pas nécessairement une solution optimale (algorithme d'approximation, heuristique, et plus particulièrement ici, algorithmes génétiques).

Très grossièrement, un algorithme génétique débute avec un ensemble de solutions valides (la population) qui peuvent, par exemple, être arbitraires ou aléatoires. C'est la génération 1. À la génération  $i \geq 1$ , on dispose d'une population de solutions valides qui va évoluer de la façon suivante. Tout d'abord, les individus de la population courante sont "croisés" : par exemple on prend 2 solutions (ou plus) courantes (i.e., 2 itinéraires pour notre VRP) est on les "combine" pour obtenir une nouvelle solution valide (comme le croisement de 2 chromosomes dans la reproduction). On peut également modifier aléatoirement chaque nouvel individu. Par exemple, si un cycle courant utilise les arêtes  $\{a, b\}$  et  $\{c, d\}$ , on peut le modifier en "inversant" ces arêtes, c'est-à-dire utiliser les arêtes  $\{a, c\}$  et  $\{b, d\}$ . Cela correspond aux mutations des chromosomes. Enfin, on ne garde que les meilleures solutions qui constitueront les individus de la génération suivante (sélection naturelle).

Le but du projet est donc de proposer différents algorithmes génétiques pour résoudre le problème et de comparer leurs performances (longueur des solutions, temps de calcul...) entre eux et avec les algorithmes précédemment cités. Bien sûr, d'un point de vue médiation, il s'agit de vulgariser la notion d'algorithmes génétiques<sup>3</sup>.

**Notions abordées :** Algorithmes génétiques, Graphes, programmation dynamique, approximation.

---

3. N'étant pas moi même spécialiste des algorithmes génétiques, j'attends d'apprendre des choses :)

## 5 Arbre de Steiner et bulles de savon

On considère un ensemble de points dans le plan, qui correspondent à des villes. Le but est de construire des routes reliant toutes les villes, tout en minimisant la longueur totale des routes. Informellement, le but est de tracer des lignes dans le plan (de longueur totale minimum) de sorte qu'on puisse ensuite aller de n'importe quel point à n'importe quel autre en suivant ces lignes. Par exemple, si il n'y a que 2 points, la solution optimale est évidemment (?) le segment de droite entre ces deux points. Des exemples avec 4 points sont donnés à la figure 3.

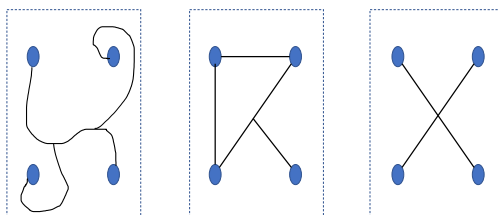


FIGURE 3 – Trois solutions valides pour un ensemble de 4 points (en bleu). De gauche à droite, la longueur de la solution semble diminuer. Est-ce que la solution de droite est optimale ?

On pourra commencer par proposer une interface qui permette aux utilisateurs de placer des points et de proposer une solution (dont la longueur sera évaluée). Ensuite, vous implémenterez des algorithmes pour déterminer une solution optimale dans le cas de 2 ou 3 points (voir les vidéos d'Olivier Druet ci-dessous). Essayez d'aller plus loin (plus de points). En particulier, on attend un algorithme qui calcule une solution (même si pas optimale) pour 4 points.

Le cas du plan est compliqué, on veut donc se restreindre à un cas plus "simple" : les graphes. Soient un graphe  $G = (V, E)$  avec une fonction de poids  $\omega$  sur les arêtes et un ensemble  $T \subseteq V$  de sommets (appelés *terminaux*) que l'on veut connecter. Le but est de trouver un sous graphe (appelé *arbre de Steiner*) de  $G$  de poids minimum et qui connecte tous les terminaux. Mathématiquement :

**Entrées :** un graphe  $G = (V, E)$ ,  $\omega : E \rightarrow \mathbb{R}^+$  et  $T \subseteq V$ .

**Sortie :** un sous graphe connexe  $H = (V', E')$  de  $G$ , avec  $T \subseteq V'$  et tel que  $\omega(H) = \sum_{e \in E'} \omega(e)$  est minimum.

Il s'agit d'un problème NP-complet. Votre application devra proposer des algorithmes qui calculent des solutions optimales et des algorithmes d'approximation pour ce problème. Voir la page Wikipedia [https://fr.wikipedia.org/wiki/Probl%C3%A8me\\_de\\_l'arbre\\_de\\_Steiner](https://fr.wikipedia.org/wiki/Probl%C3%A8me_de_l'arbre_de_Steiner).

Enfin, ces problèmes ont un lien surprenant/inattendu (?) avec la physique (en l'occurrence, les films de savon). D'un point de vue médiation, il sera intéressant d'expliquer l'interaction entre physique, mathématique et informatique. Par exemple, comment des solutions issues de la physique permettent d'aider les mathématiciens et vice versa ? Voir les vidéos d'Olivier Druet.

<https://www.youtube.com/watch?v=YsuhZ61zvLo>

<https://www.youtube.com/watch?v=E05bgsn2ips>

C'est une bonne occasion de proposer une activité débranchée pour concilier physique, mathématique et informatique.

**Notions abordées :** Géométrie, graphes, algorithme exponentiel, approximation, physique

## 6 Jeu de Hex et au delà

**Jeux positionnels.** Commençons par un jeu classique. Au morpion, deux joueurs jouent sur une grille  $3 \times 3$ , où ils sélectionnent alternativement des cases. Le premier joueur gagne si il parvient à sélectionner toutes les cases d'une ligne, d'une colonne ou d'une diagonale. Le second joueur gagne dans le cas contraire.

Pour généraliser ce jeu, définissons tout d'abord les hypergraphes. Un hypergraphe  $H = (V, E)$  est la donnée d'un ensemble de sommets  $V$  et d'un ensemble d'hyper-arêtes  $H$ , avec chaque hyper-arête est une ensemble de sommets, i.e., pour tout  $e \in E$ ,  $e \subseteq V$ . Les hypergraphes généralisent les graphes dans le sens que, dans un graphe, chaque (hyper)arête a cardinalité 2 (une arête est un ensemble de "seulement" 2 sommets, une hyper-arête peut contenir plus que 2 sommets).

On peut maintenant donner la définition générale d'un jeu positionnel (version Maker-Breaker). Le plateau de jeu est un hypergraphe  $H = (V, E)$ . Deux joueurs, Alice et Bob, s'affrontent. Chaque joueur, tout à tour (en commençant par Alice), sélectionne un sommet qui n'a pas déjà été sélectionné. Pour visualiser, on peut dire que quand Alice sélectionne un sommet, elle le marque d'un symbole (e.g., une croix, ou elle le colore en rouge) et lorsque Bob sélectionne un sommet, il le marque d'un autre symbole (e.g., un rond, ou il le colore en bleu). Le jeu se termine lorsque tous les sommets ont été sélectionnés. Si Alice a réussi à sélectionner tous les sommets d'une hyper-arête, elle gagne. Sinon, Bob gagne. C'est-à-dire qu'une hyper-arête correspond à un ensemble gagnant pour Alice. Voir (au moins l'introduction de) <http://www-sop.inria.fr/members/Nicolas.Nisse/slides/LAWCG24.pdf> pour d'autres exemples.

Dans ce type de jeux combinatoires, la question est de savoir qui de Alice ou Bob a une stratégie gagnante (une stratégie qui permette de gagner quelles que soient les réactions de l'adversaire).

**Jeu de Hex.** Ce jeu se joue sur une grille avec des cases hexagonales telle que deux côtés opposés de la grille sont colorés, disons rouge. À chaque tour, Alice et Bob alternativement choisissent une case non colorée et la colore (d'abord Alice en rouge et ensuite Bob en bleu). Alice gagne si elle crée un chemin rouge liant les deux côtés rouges. Bob gagne s'il l'en empêche<sup>4</sup>. Voir <https://fr.wikipedia.org/wiki/Hex> et surtout <https://www.youtube.com/watch?v=NL9imSjJi7o> et les vidéos suivantes.

Ce jeu a ceci de remarquable que l'on peut prouver que Alice a toujours une stratégie gagnante mais qu'on ne la connaît que dans le cas de "petites" grilles. Saurez vous vulgariser le fait qu'en maths, on peut prouver l'existence de solutions sans les connaître explicitement ?

On commencera par proposer une interface qui permette aux utilisateurs de choisir la taille de la grille et de jouer (avec 2 joueurs humains) et de déterminer la victoire de l'un des joueurs. Ensuite, on veut un algorithme qui puisse jouer (le mieux possible) contre un humain. Pour cela, un algorithme de type Min-Max est attendu ([https://fr.wikipedia.org/wiki/Algorithme\\_minimax](https://fr.wikipedia.org/wiki/Algorithme_minimax)).

### Aller plus loin ?

- On peut généraliser Hex avec un graphe quelconque (pas seulement une grille hexagonale). Étant donné un graphe  $G = (V, E)$  et deux sommets  $u$  et  $u'$  initialement colorés rouge.

---

4. L'hypergraphe associé a donc un sommet par case (un sommet interne a 6 voisins) et l'ensemble des hyper-arête sont les chemins qui relient les côtés rouges.

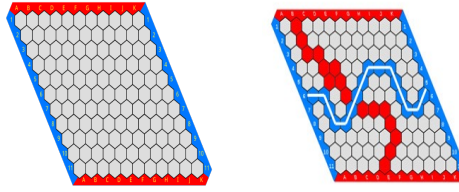


FIGURE 4 – Exemple d’un jeu de Hex sur une grille hexagonale  $11 \times 11$ . À droite, une partie où Bob a gagné puisqu’il a créé un chemin bleu empêchant Alice de relier les 2 côtés rouges.

Alice et Bob colorent alternativement les sommets non colorés comme dans le jeu de Hex. Est-ce que Alice peut créer un chemin rouge liant  $u$  à  $u'$  ?

- Étant donné un graphe  $G = (V, E)$ , on peut définir les ensembles gagnants comme  $\{N(v) \cup \{v\} \mid v \in V\}$  avec  $N(v)$  l’ensemble des voisins de  $v$ . C’est-à-dire que, pour gagner, Alice doit colorer rouge un sommet  $v$  ainsi que tous ses voisins. Quand Alice a-t-elle une stratégie gagnante ? Ce jeu est connu comme le *domination game*.
- En général, les jeux positionnels Maker-Breaker sur un hypergraphe<sup>5</sup> sont PSPACE-complets lorsque les hyper-arêtes ont taille 4 [Galliot 2025] et polynomiaux si les hyper-arêtes sont de taille 3 [Galliot et al. 2023]. Saurez vous vulgariser ces concepts (polynomial, NP, NP-dur, PSPACE...) de complexité ?

**Notions abordées :** Jeux combinatoires, stratégie gagnante, Hex, PSPACE...

---

5. Étant donnée une instance du jeu, le problème de décision est de savoir qui de Alice ou Bob a une stratégie gagnante.

## 7 Ronde des couleurs et taquin

**Ronde des couleurs.** On considère un ensemble de  $n$  “bases” organisées en cercle et telles que chaque base a sa propre couleur. Mathématiquement, on a un cycle  $(0, \dots, n-1)$  avec  $c_i$  la couleur de la base  $i$  et  $c_i \neq c_j$  pour tout  $i \neq j$  (deux bases distinctes ont des couleurs différentes). Pour chaque  $0 < i < n$ , on a  $x \geq 2$  briques de couleurs  $c_i$  et on a  $x-1$  briques de couleur  $c_0$ . Donc on dispose d’un ensemble de  $nx-1$  briques au total. Initialement, on dispose arbitrairement les briques dans les bases tel que chaque base  $i > 0$  a  $x$  briques sauf une base qui n’a que  $x-1$  briques. Une base ne peut toujours contenir que au plus  $x$  briques. À un tour, supposons que la base  $0 \leq j < n$  ne contient que  $x-1$  briques. Alors, on peut déplacer une brique de la base  $j-1 \pmod n$  vers la base  $j$ , ou alors, on peut déplacer une brique de la base  $j+1 \pmod n$  vers la base  $j$ . Le but est d’atteindre une configuration où chaque base de contient que des briques de sa couleur. Voir <https://www.youtube.com/watch?v=UG0yKWZ06PA>.



FIGURE 5 – Exemple avec  $n = 5$ ,  $x = 2$  et la couleur  $c_0$  est la couleur verte. Initialement, la base jaune n’a qu’une brique (ici un carré de couleur violette) et les autres bases en ont deux.

Commencez par une application qui permette à un joueur humain de jouer au jeu précédent (donc vérifiant la validité des coups, et déterminant la condition de victoire). Puis, concevez un algorithme qui gagne à tous les coups (quelle que soit la configuration de départ).

Le jeu précédent est une version à une dimension du jeu classique du taquin. Saurez vous expliquer simplement pourquoi? Le jeu de la ronde des couleurs (à une dimension) est toujours possible, alors que le taquin (“deux dimensions”) n’est pas forcément réalisable selon la configuration de départ. Saurez vous expliquer simplement pourquoi?

**Taquin.** Étant donnée une grille  $n \times m$  avec des cases étiquetées de 1 à  $n * m - 1$  et une case vide étiquetée 0. À chaque étape, on peut échanger la position de la case 0 avec celle de l’une de ses (au plus) quatre cases voisines. Le but est de déterminer un algorithme qui permet de déplacer (s’il c’est possible) les cases depuis n’importe quelle configuration initiale jusqu’à la configuration 1, 2, 3,  $\dots$ ,  $mn - 1, 0$  de gauche à droite et de haut en bas.

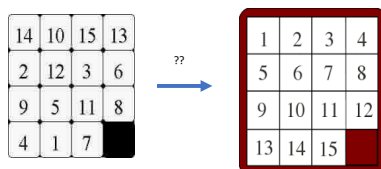


FIGURE 6 – Exemple d’un jeu de taquin  $4 \times 4$ .

On veut ici une application qui génère un jeu de taquin (par exemple avec des images plutôt que des nombres), permette à un humain d'y jouer et aussi propose un algorithme qui résolve le problème (ou argumente sur le fait que la résolution n'est pas possible).

Une vidéo de Micmaths (Mickaël Launey) :

<https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.youtube.com/watch%3Fv%3D-3IsCOJieCc&ved=2ahUKEwi6-8nd5s-LAxXPSPEDHWfkPI4QtwJ6BAGMEAI&usg=A0vVaw0bXjHkYptm8C5VWZvc0kgM>

Une vidéo de Numberphile :

<https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.youtube.com/watch%3Fv%3DYI1WqYKHi78&ved=2ahUKEwium7KA58-LAxW4VPEDHYZ7HSwQtwJ6BAGKEAI&usg=A0vVaw1qxpH17l1z4ffDN6jGggS>

**Notions abordées :** Preuve d'impossibilité, algorithme glouton, recherche exhaustive, applications...

## 8 Jeu des bâtonnets, apprentissage par renforcement et probabilités

**Le jeu classique.** Prenons l'exemple classique du jeu de Fort Boyard : il s'agit d'un jeu à deux joueurs qui jouent tour-à-tour. On dispose initialement d'un certain nombre  $x$  de bâtonnets et d'un ensemble  $\mathcal{M}$  de coups possibles. À son tour, le joueur courant peut retirer  $y \in \mathcal{M}$  bâtonnets (à Fort Boyard,  $\mathcal{M} = \{1, 2, 3\}$ , donc à son tour, le joueur peut retirer 1, 2 ou 3 bâtonnets). Dans le jeu de Fort Boyard, le joueur qui prend le dernier bâtonnet (donc s'il ne reste qu'un bâtonnet lorsque c'est son tour) perd (version misère).

Ici, on considère une version plus générale du jeu : on commence avec  $x$  bâtonnets ; à chaque tour, le joueur courant peut retirer  $y \in \mathcal{M}$  bâtonnets (pour  $\mathcal{M}$  un ensemble d'entiers donné) et le vainqueur est le joueur qui retire le/les dernier(s) bâtonnet(s) (version normale).

On commencera par proposer une interface qui permette aux utilisateurs de choisir le nombre initial  $x$  de bâtonnets et l'ensemble  $\mathcal{M}$  de coups possibles et aussi la condition de victoire (normale ou misère) et de jouer (avec 2 joueurs humains) et de déterminer la victoire de l'un des joueurs. Ensuite, on veut un algorithme qui puisse jouer (le mieux possible) contre un humain. Pour cela, voir Chapitre 7 de <https://hal.science/hal-03787593>.

**Apprentissage par renforcement.** Étant donné un tel jeu  $(x, \mathcal{M})$ , il est facile de décrire un algorithme (d'apprentissage par renforcement) qui apprenne à jouer. Voir [https://portail.terra-numerica.org/media/Deroule\\_atelier\\_la\\_machine\\_qui\\_apprend\\_a\\_jouer\\_au\\_nim\\_KBnn0J9.pdf](https://portail.terra-numerica.org/media/Deroule_atelier_la_machine_qui_apprend_a_jouer_au_nim_KBnn0J9.pdf)

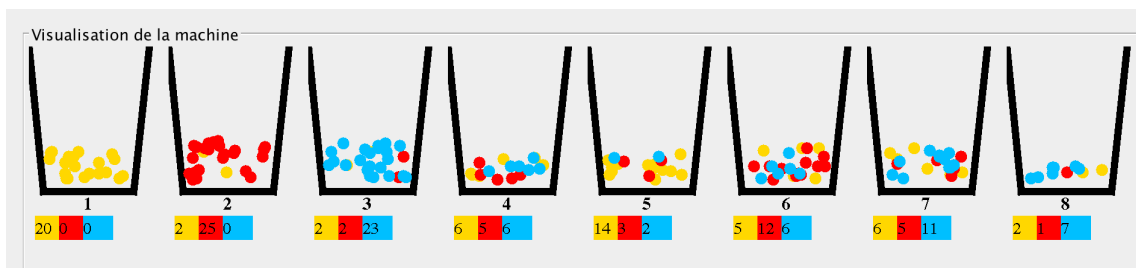


FIGURE 7 – Apprentissage par renforcement du jeu des bâtonnets.

Sans rentrer dans les détails, l'algorithme joue initialement au hasard : pour chaque position du jeu (nombre de bâtonnets restants), tous les coups de  $\mathcal{M}$  sont équiprobables. À la fin de chaque partie, si l'algorithme a gagné, les probabilités sont mises à jour pour favoriser les coups qui ont mener à la victoire. Si, au contraire, l'algorithme perd, les probabilités des coups qui ont menés à la défaite sont diminués. Il s'agira d'implémenter cet apprentissage avec divers paramètres : le nombre de bâtonnets, l'ensemble  $\mathcal{M}$ , la façon de mettre à jour les probabilités... Voir un exemple avec l'application d'Eric Duchêne : <https://projet.liris.cnrs.fr/~mam/machine/>.

Nous sommes particulièrement intéressé par le temps de convergence de l'apprentissage en fonction des différents paramètres. Il faudra donc faire des campagnes de tests pour obtenir des statistiques sur le temps de convergence. Id éalement, on aimerait également des bornes théoriques. Notons qu'il s'agira d'abord de bien définir ce qu'on entend par "temps de convergence".

**Notions abordées :** Probabilités, Jeux combinatoires, stratégie gagnante, apprentissage par renforcement...

## 9 Reconnaissance de chiffres et réseaux de neurones

Nos collègues de la MMI ont construit une excellente activité débranchée pour expliquer ce qu'est un réseau de neurones et comment il peut être utilisé pour reconnaître des chiffres. Voir <https://mmi.universite-lyon.fr/pour-les-scolaires/ressources-pedagogiques-et-de-mediation/connecte-tes-neurones-385696.kjsp>.

Pour cela, ils ont judicieusement proposé un réseau de neurones et un ensemble d'instances (des chiffres dessinés qui doivent être reconnus) de sorte que : soit le chiffre en entrée est effectivement reconnu, soit une simple (et surtout rapide) modification des paramètres (par rétropropagation de gradient) permet de reconnaître le chiffre.

**Dans un premier temps, il s'agit de transposer leur activité en une application online.**

Puis, il s'agira de proposer une méthode systématique qui construise des instances qui, ou bien soient reconnues efficacement, ou bien ne nécessitent qu'un nombre limité d'itérations (par rétropropagation de gradient) pour modifier les paramètres de façon à être reconnues.

### Aller plus loin ?

- Proposer une autre activité débranchée, voir <https://www.youtube.com/watch?v=rA5qnZUXcqo>.
- Une fois les réseaux de neurones bien compris, il est possible d'aborder les LLM. Saurez-vous vulgariser le fonctionnement des IA génératives ?

**Notions abordées :** Reconnaissance d'image, réseau de neurones, rétropropagation de gradient, apprentissage par renforcement, LLM...