REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'enseignement supérieur et de la recherche scientifique

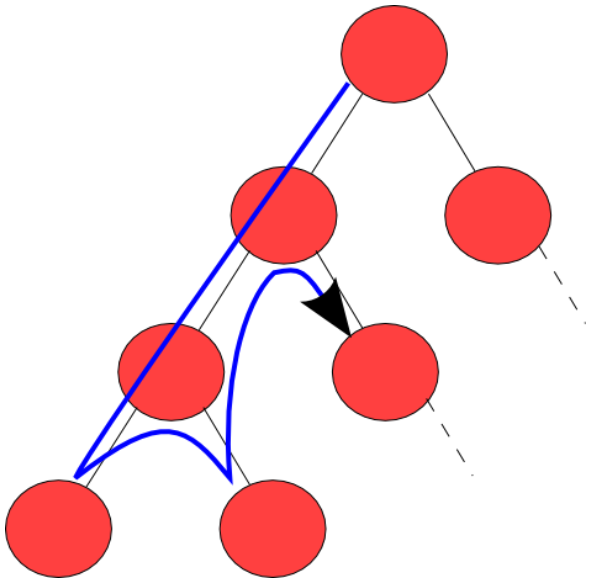
Université Larbi Ben M'Hidi Oum El Bouaghi

Faculté des sciences exactes et science de la nature et de la vie

Département de mathématique d'informatique et

Cours : Méthodes Algorithmiques

Chapitre 4 : Backtracking



Proposé par Dr. Kenza Belhouchette

Edition 1.0

2024/ 2025

Introduction

Le **Backtracking** est une technique algorithmique essentielle et polyvalente, largement utilisée dans la résolution de problèmes de nature combinatoire et dans l'optimisation. Cette approche repose sur une exploration organisée de toutes les solutions possibles, souvent visualisée sous forme d'un arbre de décision. Dans cet arbre, chaque étape correspond à une décision qui conduit à plusieurs choix, explorés tour à tour pour évaluer leur conformité aux contraintes du problème.

À la différence des méthodes optimisées comme les approches gloutonnes ou la programmation dynamique, qui cherchent à éviter l'exploration exhaustive en ciblant des solutions optimales ou quasi-optimales dès les premières étapes, le backtracking se base sur une stratégie d'essai-erreur. Lorsqu'une direction dans l'arbre ne mène pas à une solution viable, l'algorithme revient en arrière et explore d'autres possibilités. Cette capacité à "reculer" permet d'identifier une solution optimale ou l'ensemble des solutions valides, mais elle peut se révéler coûteuse en termes de ressources informatiques.

Le backtracking s'applique à de nombreux problèmes classiques, tels que le problème des N reines sur un échiquier, la résolution de casse-têtes comme le Sudoku, ou encore le coloriage de graphes. Ces exemples démontrent comment cette méthode permet de naviguer dans un espace de solutions tout en intégrant des techniques d'optimisation, comme la réduction de l'espace de recherche (pruning), afin d'éviter les calculs inutiles.

Ce chapitre aborde le backtracking en profondeur, en détaillant ses concepts de base, sa structure algorithmique et ses limites. Nous présenterons aussi des exemples pratiques et des exercices viendront illustrer cette approche, offrant ainsi aux étudiants des outils pour mieux comprendre son utilisation dans différents contextes.

Définition et Concept du Backtracking

Le **Backtracking** est une approche algorithmique basée sur l'exploration systématique d'un ensemble de solutions possibles dans le but de résoudre des problèmes de décision, d'optimisation ou de satisfaction de contraintes. Il s'agit d'une technique de recherche exhaustive qui fonctionne en construisant les solutions de manière incrémentale, en ajoutant une décision à la fois, et en annulant ("backtrack") les décisions antérieures si elles ne mènent pas à une solution valide ou optimale.

Dans le cadre théorique, le backtracking peut être représenté par un **espace de recherche** sous forme d'un **arbre de décision** où chaque nœud correspond à un état partiel de la solution et chaque branche représente un choix (ou une décision) permettant d'étendre cet état. En pratique, on part d'un état initial et l'on parcourt récursivement cet arbre, en suivant un parcours en profondeur (**Depth-First Search**).

Processus Mathématique

Considérons un problème où l'on cherche une solution S qui satisfait une certaine condition C(S).

Soit Décisions=$\{d_1,d_2,\ldots,d_k\}$l'ensemble de toutes les décisions possibles à chaque étape.

Le backtracking tente de construire S en partant d'un état vide ∅, en ajoutant successivement des décisions d_i pour progresser vers un état complet.

Formellement, si S=$\{d_{i1},d_{i2},\ldots,d_{im}\}$est une solution potentielle en construction, à chaque étape t, l'algorithme :

1. Ajoute un choix d_{it} à S s'il satisfait les contraintes intermédiaires $C(S_t)$.
2. Si un choix mène à une impasse, c'est-à-dire qu'aucune extension de l'état actuel ne permet de satisfaire C(S), alors l'algorithme effectue un **backtrack** en annulant le dernier choix, revenant ainsi à l'état précédent pour explorer d'autres choix disponibles.

Cette méthode se base donc sur un **parcours en profondeur récursif** de l'arbre, où l'algorithme ajoute et retire les choix de manière dynamique.

Optimisation et Réduction de l'Espace de Recherche

Dans le cadre du backtracking, l'optimisation de l'espace de recherche est cruciale pour limiter la complexité de calcul, qui, en l'absence de contraintes, croît de manière exponentielle. Pour un problème combinatoire avec n décisions possibles à chaque étape sur une profondeur de m, l'arbre de recherche pourrait théoriquement contenir jusqu'à $O(n^m)$ solutions partielles.

Pour restreindre cet espace, le backtracking s'appuie sur des **fonctions de contraintes** qui éliminent les branches non valides au cours de l'exploration. Lorsqu'une décision ajoutée enfreint les conditions du problème, l'algorithme interrompt l'exploration de cette branche, ce qui est communément appelé **élagage** ou **pruning**. Mathématiquement, cela se traduit par une réduction du nombre de nœuds explorés, car seules les branches répondant aux critères de validité sont développées. Cette technique permet de limiter la complexité de l'algorithme en supprimant des sous-arbres non prometteurs.

Considérons, par exemple, un problème de coloration de graphe où chaque sommet doit être colorié de manière à ce qu'aucun sommet adjacent n'ait la même couleur. Ici, à chaque décision (choix de couleur pour un sommet), une vérification de contrainte peut être effectuée : si un choix de couleur mène à un conflit, la branche correspondante est immédiatement éliminée, évitant ainsi toute exploration inutile de solutions incompatibles.

Le pruning permet donc de diminuer l'espace de recherche de manière drastique, ce qui a un impact positif sur la complexité en temps et en mémoire, et améliore la performance de l'algorithme dans les cas pratiques.

Principe de Fonctionnement du Backtracking

Le backtracking repose sur une exploration systématique et récursive de toutes les solutions possibles à un problème, en construisant progressivement une solution partielle, étape par étape, jusqu'à atteindre une solution complète ou déterminer qu'aucune solution ne peut être trouvée dans la branche en cours. Le principe de base consiste à avancer dans l'espace de solutions en ajoutant des choix successifs, et à revenir en arrière (backtrack) si une configuration partielle ne mène pas à une solution valide.

1. **Construction Incrémentale** : L'algorithme démarre avec une solution initiale vide ou partiellement construite. À chaque étape, une nouvelle décision est ajoutée à la solution actuelle. Cette décision doit être compatible avec les contraintes définies par le problème, appelées ici **conditions de validité**.
2. **Vérification des Contraintes** : Après chaque ajout, l'algorithme vérifie si la solution partielle respecte les contraintes spécifiques du problème. Si les contraintes sont satisfaites, l'algorithme poursuit en ajoutant une nouvelle décision. Si, au contraire, une contrainte est violée, l'algorithme abandonne cette branche de recherche.
3. **Retour en Arrière (Backtrack)** : Si une solution partielle ne peut pas être étendue pour former une solution valide, l'algorithme effectue un retour en arrière, supprimant le dernier choix fait et essayant une autre option parmi celles encore disponibles. Ce mécanisme de backtrack permet de "nettoyer" les choix précédents pour explorer des alternatives, assurant ainsi que toutes les combinaisons possibles sont couvertes sans redondance inutile.
4. **Terminaison** : Le processus se termine lorsque toutes les branches de l'arbre de recherche ont été explorées ou lorsqu'une solution satisfaisant toutes les contraintes a été trouvée. Dans le cas d'un problème de satisfaction, où plusieurs solutions peuvent exister, l'algorithme peut être ajusté pour continuer la recherche jusqu'à identifier toutes les solutions possibles.

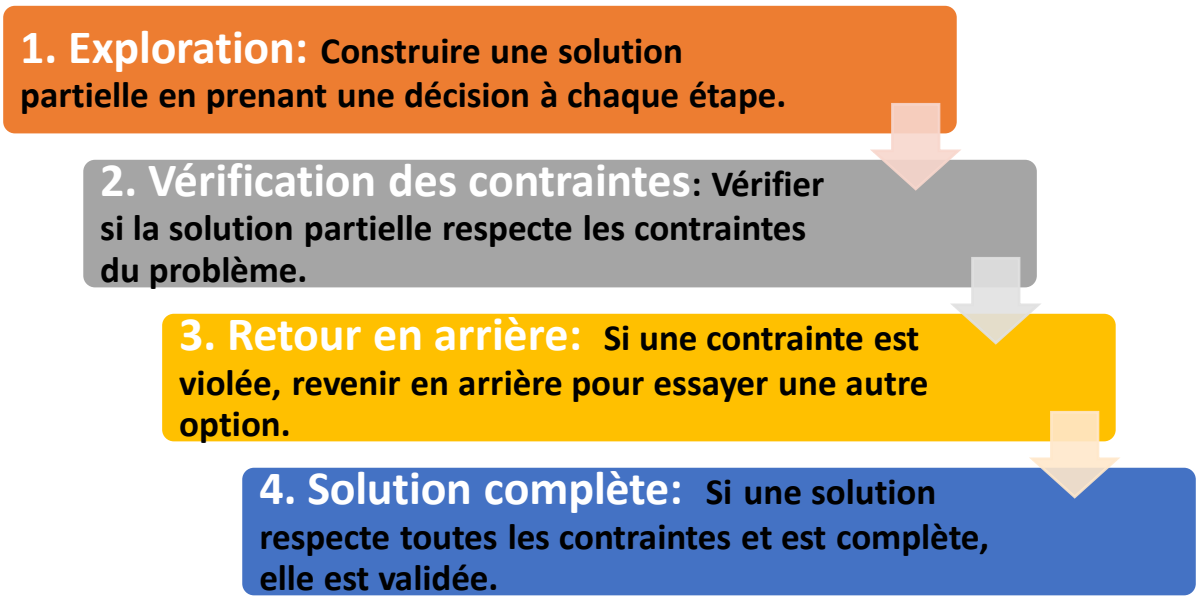


Figure 1. Étapes Principales du Backtracking

Structure Générale d'un Algorithme de Backtracking

Un algorithme de backtracking peut être décrit de manière structurée en plusieurs étapes systématiques. Ce type d'algorithme s'articule autour d'un schéma de construction et de vérification récursif, permettant l'exploration complète des solutions possibles tout en éliminant les chemins non valides. Voici la structure détaillée d'un algorithme de backtracking :

1. **Initialisation** : On définit les paramètres nécessaires pour le problème, notamment l'ensemble des choix possibles, les contraintes, et une structure pour stocker la solution partielle en cours de construction.
2. **Fonction Principale de Backtracking** : Cette fonction est au cœur de l'algorithme et prend en paramètre l'état actuel de la solution partielle ainsi que le niveau ou l'étape en cours de traitement dans l'arbre de décision. Cette fonction se charge d'explorer toutes les options disponibles pour le problème donné en suivant un parcours récursif.
3. **Vérification des Contraintes** : Pour chaque choix ajouté à la solution partielle, l'algorithme vérifie si ce choix respecte les contraintes spécifiques du problème. Si les contraintes sont satisfaites, l'algorithme poursuit l'exploration ; sinon, il abandonne cette branche, réduisant ainsi l'espace de recherche. Cette vérification est essentielle pour optimiser la recherche et éviter les branches sans issue.
4. **Ajout d'un Choix (Étape de Décision)** : Si les contraintes sont vérifiées, l'algorithme ajoute le choix actuel à la solution partielle et continue l'exploration en passant à l'étape suivante (ou niveau suivant) dans l'arbre de décision. À ce stade, l'algorithme tente d'étendre la solution partielle en choisissant la prochaine option parmi celles disponibles.
5. **Retour en Arrière (Backtrack)** : Si aucune solution valide n'est possible en partant de la solution partielle actuelle, l'algorithme retourne en arrière pour annuler le dernier choix ajouté. Cela permet d'explorer d'autres possibilités sans redondance, en essayant de nouvelles combinaisons qui pourraient aboutir à une solution valide.
6. **Condition de Terminaison** : Le processus de backtracking se termine lorsqu'une solution complète satisfaisant toutes les contraintes est trouvée ou lorsque toutes les possibilités ont été explorées et aucune solution n'a été trouvée. Pour des problèmes qui nécessitent toutes les solutions possibles, l'algorithme peut continuer l'exploration jusqu'à l'examen complet de toutes les branches de l'arbre.

Schéma d'un Algorithme de Backtracking

```
fonction Backtracking(état_actuel, niveau) :
    si niveau == dernier_niveau alors
        si solution_valide(état_actuel) alors
            enregistrer_solution(état_actuel)
        fin si
        retourner
    fin si
    pour chaque choix dans choix_possibles(niveau) :
        si contrainte_respectée(état_actuel, choix) alors
            ajouter(état_actuel, choix)
            Backtracking(état_actuel, niveau + 1)
            retirer(état_actuel, choix)
        fin si
    fin pour
```

- **Point de départ** : La fonction Backtracking commence par examiner l'état actuel de la solution et le niveau de profondeur dans l'arbre de décision.
- **Condition de terminaison** : Si le niveau atteint le dernier niveau (la profondeur maximale de l'arbre de décision), l'algorithme vérifie si l'état actuel forme une solution valide. Si c'est le cas, la solution est enregistrée.
- **Exploration des choix possibles** : Si l'algorithme n'a pas atteint le dernier niveau, il explore chaque choix possible pour l'état actuel à ce niveau spécifique.

- **Vérification des contraintes** : Avant d'ajouter un choix à la solution partielle, l'algorithme vérifie que ce choix respecte toutes les contraintes du problème.
- **Construction de la solution** : Si la contrainte est respectée, le choix est ajouté à la solution partielle et la fonction récursive est appelée pour explorer la suite des décisions.
- **Retour en arrière** : Après avoir exploré un chemin (sous-arbre) et ajouté un choix, l'algorithme revient sur le dernier choix en le retirant et en continuant l'exploration des autres alternatives possibles.

Comparaison entre le Backtracking et la Programmation Dynamique

Le backtracking et la programmation dynamique sont deux approches algorithmique utilisées pour résoudre des problèmes combinatoires ou d'optimisation, mais elles diffèrent considérablement en termes de fonctionnement, d'efficacité et d'application.

Caractéristiques	Backtracking	la Programmation Dynamique
Complexité et Efficacité	Complexité souvent **exponentielle**, car il explore toutes les solutions possibles (dans le pire des cas).Peut être optimisé avec des techniques d'élagage (pruning), mais reste inefficace pour les grands espaces de solutions.	Complexité généralement **polynomiale**, car elle évite les calculs redondants.Elle est particulièrement efficace lorsque le problème présente une **structure de recouvrement des sous-problèmes** et une **optimalité des sous-structures**.
Applications Typiques	▪ Résolution de puzzles (ex. Sudoku, problème des N-reines). ▪ Problèmes combinatoires (ex. génération de permutations, arbre de décision). ▪ Recherche exhaustive avec des contraintes complexes.	▪ Problèmes d'optimisation (ex. problème du sac à dos, plus longue sous-séquence commune). ▪ Problèmes sur les graphes (ex. plus court chemin avec Bellman-Ford). ▪ Problèmes récurrents (ex. découpage de chaînes ou matrices, alignement de séquences).
Utilisation de la Mémoire	▪ Utilise généralement une pile d'appels récursive pour stocker les états partiels de la solution. ▪ Faible consommation mémoire dans sa version de base.	▪ Nécessite une table de mémorisation (tableau ou dictionnaire) pour stocker les résultats intermédiaires. ▪ Consommation mémoire plus élevée, mais avec une optimisation possible

		(ex. réduction en espace pour les algorithmes linéaires).
Avantage principal	▪ Simple à implémenter pour explorer toutes les solutions.	▪ Très efficace pour les problèmes d'optimisation.
Inconvénient majeur	Très lent pour les grands problèmes (complexité exponentielle).	Nécessite de reconnaître les sous-problèmes pour l'implémentation.
Adaptabilité	Convient mieux aux problèmes avec peu de contraintes.	Plus adapté aux problèmes ayant une structure répétitive.

Problèmes Classiques Résolus par Backtracking

L'algorithme de backtracking est utilisé pour résoudre une large gamme de problèmes combinatoires, notamment ceux où une solution doit être construite étape par étape, en vérifiant les contraintes à chaque niveau de la construction. Voici quelques exemples classiques de problèmes résolus efficacement par cette méthode :

1. Le Problème des N-Reines

Le problème des N-reines consiste à placer N reines sur un échiquier de taille N × N de manière à ce qu'aucune reine ne puisse en attaquer une autre. Cela implique que les reines doivent être placées de sorte qu'elles ne partagent pas la même ligne, colonne ou diagonale.

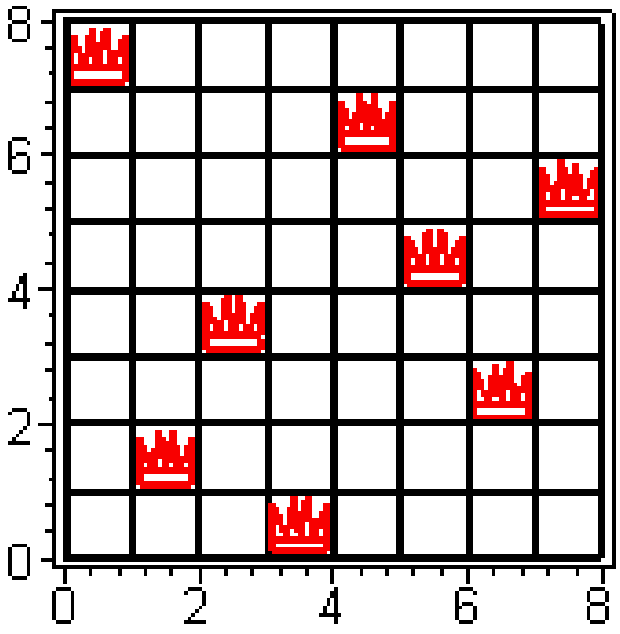


Figure 2.Le Problème des N-Reines

Définition Formelle

Données :
N, la taille de l'échiquier (N×N).

Contraintes :

1. Une seule reine par ligne.
2. Une seule reine par colonne.
3. Aucune reine ne doit partager une diagonale.

Objectif : Trouver toutes les configurations possibles ou une seule configuration valide.

Approche Backtracking :

À chaque étape, l'algorithme tente de placer une reine dans une colonne, puis vérifie si ce placement est valide en tenant compte des contraintes (pas de reine sur la même ligne, colonne ou diagonale). Si un placement est valide, l'algorithme passe à la colonne suivante. En cas d'impasse (lorsque toutes les lignes sont occupées dans une colonne donnée), il revient en arrière et modifie le dernier placement.

Étapes principales :

1. **Placer une reine** sur une ligne donnée.
2. **Vérifier les contraintes** : Si la position est valide (aucune reine ne peut être attaquée), continuer avec la ligne suivante.
3. **Retour en arrière** : Si aucune position valide n'est trouvée, retirer la dernière reine placée et essayer une autre possibilité.
4. **Condition de terminaison** : Une solution est trouvée lorsque N reines sont placées correctement, ou lorsque toutes les options ont été explorées.

Modélisation sous forme d'Arbre de Décision

Le problème des N-Reines peut être visualisé comme un **arbre de décision** :

- Chaque **niveau** de l'arbre correspond à une ligne de l'échiquier.
- Chaque **branche** représente un choix de colonne pour placer une reine.
- Les **feuilles** de l'arbre représentent des configurations complètes (solutions ou échecs).

Cette modélisation permet de suivre le chemin parcouru et de comprendre les retours en arrière effectués.

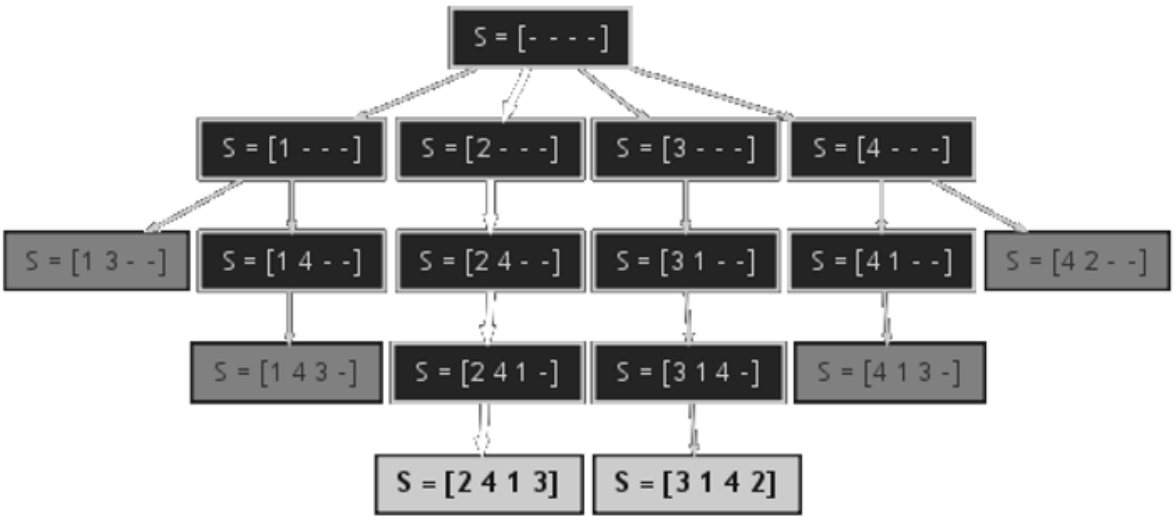


Figure.3 **N reines sous forme d'Arbre de Décision**

Pseudocode de l'Algorithme N reine en backtarcking

```
Fonction résoudre_N_reines(n):
    Créer un tableau board de taille n x n initialisé à 0
    Si placer_reine(board, 0, n) alors (Touts les reine sont placées)
        Afficher le tableau board
    Sinon
        Afficher "Pas de solution"
FinFonction
Fonction placer_reine(board, ligne, n):
    Si ligne == n alors
        Retourner Vrai //->  (toutes les reines sont placées)
    Pour col ← 0 à n-1 faire
        Si est_sûr(board, ligne, col, n) alors
            board[ligne][col] ← 1
            Si placer_reine(board, ligne + 1, n) alors
```

```
            Retourner Vrai
        FinSi
        board[ligne][col] ← 0  //-> Backtracking : retirer la reine
    FinSi
  FinPour
  Retourner Faux (impossible de placer une reine dans cette ligne)
FinFonction
Fonction est_sûr(board, ligne, col, n):
    Pour i ← 0 à ligne-1 faire
    Si board[i][col] == 1 alors //->  Vérifier la colonne
        Retourner Faux
    FinSi
  FinPour
  i ← ligne - 1, j ← col - 1
  Tant que i ≥ 0 et j ≥ 0 faire
    Si board[i][j] == 1 alors //->  Vérifier la diagonale gauche
        Retourner Faux
    FinSi
    i ← i - 1, j ← j - 1
  FinTantQue
  i ← ligne - 1, j ← col + 1
  Tant que i ≥ 0 et j < n faire
    Si board[i][j] == 1 alors
  //-> Vérifier la diagonale droite
        Retourner Faux
```

```
        FinSi

        i ← i - 1, j ← j + 1

    FinTantQue

    Retourner Vrai  //-> (la position est sûre)

FinFonction
```

Complexité

Espace de recherche : N!, correspondant à toutes les permutations possibles des N reines.

Complexité temporelle :

- Pire cas : O(N!).
- Optimisation : La complexité réelle dépend de N et de la capacité à éliminer les branches non valides.

Conclusion

Dans ce chapitre, nous avons abordé le concept de backtracking, une approche algorithmique essentielle pour résoudre des problèmes nécessitant l'exploration systématique d'un espace de solutions tout en respectant des contraintes spécifiques. Nous avons vu comment cette méthode permet de naviguer dans un arbre de décision et de revenir en arrière lorsqu'un choix se révèle infructueux, garantissant ainsi une exploration complète tout en éliminant les branches non pertinentes. Le problème des N reines a été utilisé pour illustrer cette approche. Lors des travaux dirigés, nous étudierons d'autres applications du backtracking, comme le problème de coloriage de graphes, le labyrinthe et le problème de Sudoku, afin de renforcer la compréhension de cette méthode et d'explorer ses diverses applications dans des contextes combinatoires variés. Ce chapitre souligne ainsi non seulement l'efficacité et la flexibilité du backtracking, mais également son rôle central dans la résolution de nombreux problèmes algorithmiques classiques.