Projet LAsSy

(Learning Assistance systems for Efficient Disassembly)

Rapport de stage de 5ème année (Robotique)

Présenté par

El Morabit Waël

Promotion 2024/2025

Lieu de stage : IMT Nord Europe (CERI SN), Lille, Douai (Centre de Recherche, Plateforme 4.0)

Du 15/05/2025 au 14/10/2025

Tuteur académique (IMT) : M.Motetse

Tuteur universitaire (Polytech Dijon) : M.Lew Yan Voon / M.Fofi

Année universitaire 2024 – 2025

Remerciements

Je remercie M. Motetse, pour son accueil au sein du centre de recherche IMT Nord Europe (CERI SN) durant ces cinq mois.

Je tiens également à remercier sincèrement M. Abdous et M. Lucas, enseignants-chercheurs impliqués dans le projet LAsSy, pour m'avoir suivi sur toute la durée du stage et pour m'avoir accordé leur confiance sur ce projet. En ayant trouvé un équilibre entre autonomie et suivi, ils ont su me fournir les ressources indispensables à la réussite de cette expérience professionnelle.

Par ailleurs, je remercie M. Lew Yan Voon et M. Fofi, mes tuteurs académiques à Polytech Dijon, pour les échanges constructifs que nous avons eus durant ces mois.

Enfin, je remercie tous les professeurs, étudiants, ainsi que mes proches et personnes qui ont contribué de près ou de loin au bon déroulement de mon stage.

Table des matières

Table des figures

Introduction Générale

Dans le cadre de ma cinquième année en école d'ingénieur à Polytech Dijon, en option Robotique,

J'ai eu l'opportunité d'effectuer mon stage au sein du CERI Sciences du Numérique de l'IMT Nord Europe, à Douai. J'ai occupé le poste de stagiaire en recherche, travaillant sur un projet innovant de désassemblage automatisé et ordonnancement adaptatif dans le cadre de l'industrie 4.0.

En tant qu'étudiant en robotique, vision, IA, j'ai choisi d'orienter ce stage vers un sujet de recherche à l'intersection de l'optimisation, de la robotique/cobotique et de l'intelligence artificielle appliquée.

Ce travail s'inscrit dans le projet collaboratif LAsSy qui vise à développer une cellule cobotique capable de générer et d'adapter en temps réel des séquences de démontage de produits en fin de vie.

L'IMT Nord Europe est une grande école d'ingénieurs de l'Institut Mines-Télécom, reconnue pour ses travaux dans l'industrie du futur. Son centre de recherche CERI SN mène des activités dans les domaines de la robotique, de l'optimisation et des systèmes numériques intelligents, et dispose d'une plateforme technologique 4.0 regroupant robots collaboratifs, systèmes de vision, capteurs et moyens de prototypage avancés.

Ce sujet de stage m'a particulièrement attiré car il offre une opportunité unique de travailler sur une problématique de recherche appliquée, en lien direct avec les enjeux environnementaux, de flexibilité industrielle et d'automatisation intelligente.

Ce rapport sera principalement composé de trois parties : dans un premier temps, nous décrirons le contexte, les objectifs du projet et son domaine de recherche, puis nous détaillerons la méthodologie et les travaux employés, avant d'analyser les résultats obtenus et d'en dresser un bilan.

1. Présentation de l'institution d'accueil : IMT Nord Europe – CERI SN

Présentation Générale :

IMT Nord Europe, membre de l'Institut Mines-Télécom, est une grande école publique d'ingénieurs située dans les Hauts-de-France. Elle fait partie du premier groupe public français de Grandes Écoles d'ingénierie, qui regroupe 8 universités technologiques et 2 écoles affiliées.

L'école est placée sous la tutelle du ministère de l'Économie, des Finances et de la Souveraineté industrielle et numérique, ce qui traduit son orientation forte vers l'innovation et l'industrie.

Chaque année, l'école forme environ 2 200 étudiants, dont près de 20 % d'internationaux. Elle délivre des diplômes d'ingénieur, de master et de doctorat, et encadre plus d'une centaine de doctorants. La recherche y occupe une place importante : les laboratoires publient plus de 200 articles de rang A par an et participent à de nombreux projets de recherche nationaux et européens.

Leur mission et vision :

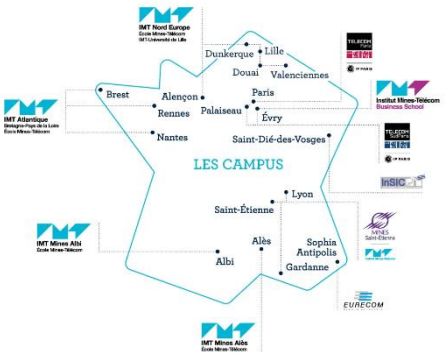
- **Pour leur mission :** Former des ingénieurs et chercheurs de haut niveau, capables de répondre aux défis de l'industrie et de la société, et contribuer à l'innovation dans les domaines de l'énergie, du numérique et de l'industrie 4.0.
- **Pour leur vision :** Se positionner comme un acteur de référence de l'industrie du futur, reconnu pour l'excellence académique et la qualité de ses partenariats industriels et internationaux.

Organisation et départements : IMT Nord Europe organise ses activités autour de plusieurs Centres d'Enseignement, de Recherche et d'Innovation (CERI), qui regroupent formation et recherche autour de thématiques clés :

- CERI Matériaux et Procédés (MP)
- CERI Énergie et Environnement (EE)
- CERI Sciences du Numérique (SN)

L'école compte environ 360 enseignants-chercheurs permanents, dont plusieurs professeurs expérimentés et encadrants de thèse.

Elle collabore aussi avec plus de 100 universités partenaires à l'international et de nombreuses entreprises à travers des projets et chaires industrielles.



"Figure 0 : Carte des campus (IMT)"

Concernant le CERI Sciences du Numérique (SN) :

Le CERI SN est le centre qui m'a accueilli pour ce stage.

Il est spécialisé dans la robotique, l'optimisation, l'intelligence artificielle, les systèmes cyber-physiques et les jumeaux numériques. Ce centre est directement impliqué dans des projets collaboratifs nationaux et européens, notamment autour de l'industrie 4.0.

Programmes et formations :

Il contribue à la formation d'ingénieurs et de doctorants autour des trois grands axes scientifiques :

- Systèmes autonomes et résilients
- Interaction, décision humaine / machine
- Modélisation et optimisation de systèmes complexes

Concernant la recherche et l'innovation :

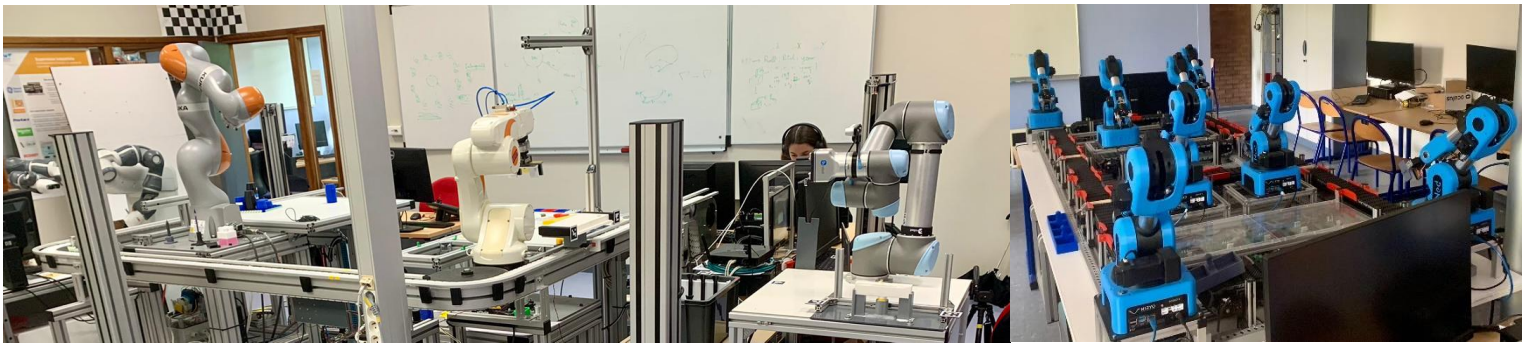
Les équipes du CERI SN mènent des recherches appliquées en lien avec l'industrie, sur des thématiques comme :

- Optimisation robuste et planification
- Intelligence artificielle et analyse de données
- Systèmes de production / industrie
- Ingénierie de systèmes
- Vision et perception artificielle pour systèmes industriels
- Réseaux / IoT
- Risques & cybersécurité
- …

Dans ce contexte large, mon travail s'inscrit surtout comme application dans le désassemblage, la planification adaptative, la robotique collaborative.

Leurs collaborations et partenariats : Il collabore avec des industriels régionaux, nationaux et internationaux, ainsi qu'avec des laboratoires académiques.

Ces partenariats permettent d'appliquer directement les résultats de recherche dans des environnements réels et d'offrir aux étudiants et doctorants l'opportunité de travailler sur des cas concrets.



"*Figure 1 :* Une partie de la Plateforme 4.0 avec robots collaboratifs"

2. Contexte et Objectif du Projet

2.1. Introduction

Le sujet choisi, intitulé « Cobotique et désassemblage en industrie 4.0 », s'inscrit dans le cadre du projet de recherche LAsSy (Learning Assistance Systems for Efficient Disassembly).

L'objectif global est donc de développer une cellule cobotique capable de démonter automatiquement des produits en fin de vie, en combinant une planification initiale et une capacité de replanification locale en cas d'imprévu.

Le projet s'inscrit dans le contexte plus large de l'économie circulaire et de l'industrie 4.0, où le désassemblage devient un enjeu majeur pour le recyclage, la réutilisation de composants et la valorisation des ressources. La planification de séquences de démontage (Disassembly Sequence Planning (DSP)) est un problème assez complexe et encore bien étudié en recherche, en particulier lorsqu'il s'agit de tenir compte des aléas (pièces bloquées, défauts, casses).

Ce rapport présente donc l'état d'avancement de mon travail après plusieurs mois de stage, avec un focus sur différents points clés.

2.2. Objectifs

Les principaux objectifs du projet étaient :

1. **Apprentissage du domaine du désassemblage automatisé**
2. **Modélisation des produits sous forme de graphes**
3. **Développement d'un pipeline de génération de séquences**
4. **Mise en place d'un mécanisme de replanification adaptative**
5. **Tests et validation de la méthode**

Pour information :

Tous ces objectifs ont été atteints au cours du stage.

Seule l'implémentation d'un scheduler fuzzy, prévue comme extension du mécanisme adaptatif, n'a pas pu être finalisée dans le temps imparti.

Ce travail se poursuivra dans le cadre du projet LAsSy par la suite.

2.3. Contexte et Problématique

Le désassemblage de produits en fin de vie est un enjeu donc central de l'économie circulaire et de l'industrie 4.0. Il permet d'extraire des composants de valeur, de séparer les matériaux pour le recyclage et de traiter les éléments dangereux en toute sécurité. Pourtant, ce processus reste difficile à automatiser en raison de la grande variabilité des produits et de leurs états de fin de vie.

La planification de séquences de désassemblage consiste elle à déterminer l'ordre dans lequel les différentes pièces doivent être retirées.

Ce problème est de nature NP-difficile : le nombre de séquences possibles croît très rapidement avec la taille du produit. De plus, une planification prédéfinie peut devenir rapidement inapplicable face aux aléas fréquents lors du démontage (pièces bloquées, endommagées, manquantes ou nécessitant une opération destructrice).

"Figure 2 : Vue éclatée d'un smartphone (exemple d'un produit à désassembler)"

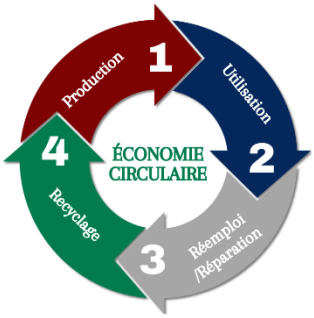
Les approches traditionnelles, basées sur des procédures fortement dépendantes de l'intervention humaine, montrent leurs limites. Elles manquent de flexibilité et de robustesse dans des environnements réels.

C'est pourquoi la recherche actuelle se concentre sur des méthodes permettant non seulement de générer un plan initial, mais aussi de s'adapter dynamiquement aux imprévus au cours de l'exécution.

Ainsi, la problématique centrale peut être résumée comme ceci :

« Comment développer un système de désassemblage capable de générer rapidement une séquence de démontage optimale ou quasi-optimale, tout en réagissant efficacement aux aléas rencontrés en cours d'exécution ? »

Cette question est au cœur de mon projet de stage, qui s'inscrit dans le développement de ses nouvelles méthodes.



"Figure 3 : Schéma du cycle de l'économie circulaire appliqué aux produits électroniques"

2.4. Contexte Scientifique et Technologique

La planification de séquences de désassemblage (DSP) a été largement étudiée dans la littérature.

Les premières approches exactes, comme la programmation linéaire en nombres entiers (MILP), garantissent une solution optimale mais ne passent pas à l'échelle dès que le nombre de composants augmente.

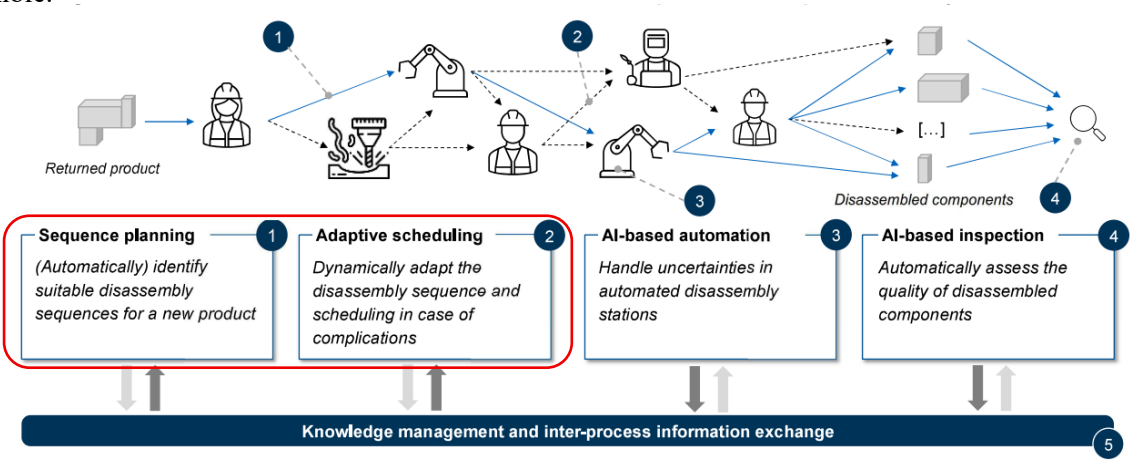
Pour dépasser ces limites, de nombreuses métaheuristiques ont été proposées, telles que les algorithmes génétiques, la recherche tabou, ou encore GRASP, offrant un bon compromis entre qualité et rapidité de calcul.

D'autres modèles comme les AND/OR graphs ou les réseaux de Pétri permettent de formaliser les dépendances logiques entre pièces, mais restent difficiles à appliquer à grande échelle.

Toutefois, la majorité de ces approches se concentrent sur la génération d'un plan initial.
Dans des environnements réels, ce plan devient souvent inapplicable face aux aléas.
La littérature récente insiste donc sur l'importance de coupler la planification à des mécanismes de replanification dynamique, capables de modifier localement la séquence en cas d'imprévu.

C'est dans ce contexte que le projet LAsSy prend forme, un seed project financé par la *German-French Academy for the Industry of the Future*, réunissant l'IMT Nord Europe (CERI SN) et la *Technische Universität München* (TUM).

Ce projet vise à explorer de nouvelles méthodes hybrides associant planification efficace et ordonnancement adaptatif, pour développer une cellule cobotique de désassemblage robuste et flexible.



"Figure 4 : Etapes du projet LAsSy complet"

En rouge (notre partie du projet)

2.5. Problématique

Le principal défi de ce projet réside dans la variabilité des produits en fin de vie et dans les aléas rencontrés lors du désassemblage.

Chaque produit peut présenter des différences selon son modèle, son état d'usure ou la présence de pièces manquantes. De plus, certaines opérations peuvent échouer en raison de pièces bloquées, endommagées ou nécessitant une destruction.

À cela s'ajoute la complexité combinatoire du problème de planification des séquences de désassemblage. Le nombre de possibilités croît rapidement avec la taille du produit, ce qui rend difficile la recherche d'une solution optimale en temps raisonnable.

Pour surmonter ces difficultés, il est nécessaire de mettre en place un plan qui combine donc :

- Une génération de séquence rapide et efficace,
- Et une capacité de replanification adaptative afin de gérer les imprévus sans repartir de zéro.

C'est précisément cette problématique qui a guidé le travail réalisé pendant ce stage, en s'appuyant sur un pipeline basé sur des algorithmes clés et sur l'ajout d'une logique de replanification locale.

2.6. Conclusion

En résumé, ce projet s'intéresse au développement du pipeline de désassemblage automatisé, capable de générer des séquences valides et de s'adapter aux imprévus rencontrés lors de l'exécution.

Les objectifs incluent non seulement la modélisation de produits sous forme de graphes, la génération de séquences par métaheuristique, mais aussi l'intégration d'une replanification adaptative pour renforcer la robustesse du système.

Les parties suivantes détailleront la méthodologie employée, les expérimentations réalisées et les résultats obtenus, afin d'offrir une vision d'ensemble complète des travaux effectués durant le stage.

3. Méthodologie

3.1. Apprentissage du domaine et préparation du projet

Avant de commencer le développement du projet, une phase d'apprentissage et de compréhension approfondie du domaine a été nécessaire.
Le sujet du désassemblage automatique et de la planification adaptative étant à la croisée de plusieurs disciplines, il a d'abord fallu s'imprégner des notions théoriques et des approches existantes avant de concevoir une méthodologie adaptée.

Durant les premières semaines et sur la continuité du stage, plusieurs réunions ont été organisées avec mes encadrants, afin de définir la portée du travail et les priorités de recherche. Ces échanges ont permis de préciser les attentes, les contraintes et les ressources disponibles.

En parallèle, une veille scientifique a été menée à partir de publications récentes sur le Disassembly Sequence Planning (DSP) et le Scheduling adaptatif.

Parmi les références étudiées figurent notamment les travaux de Guo et al. (2021) sur la modélisation du DSP, Ye et al. (2022) sur la planification flexible en environnement incertain, ainsi que plusieurs articles traitant de la modélisation graphique (AND/OR Graphs, DAG) et des stratégies de replanification (Tian, Liao, Xu, Lee…).
Ces analyses ont permis donc de mieux comprendre les tendances actuelles, les limites des approches existantes et les besoins réels de l'industrie 4.0 dans ce domaine.

À la suite de cette période d'apprentissage, un plan de travail réelle et clair a été défini et structuré autour de trois grands blocs :

1. **Modélisation du produit** : Pour représenter le produit sous forme de graphe et définir les dépendances de démontage.
2. **Planification initiale (DSP)** : Pour générer des séquences valides et efficaces.
3. **Ordonnancement adaptatif :** Pour intégrer des mécanismes de replanification et de gestion des aléas.

Cet apprentissage a été plus qu'utile pour acquérir les compétences nécessaires à la mise en œuvre du projet.

Cela a demandé un temps assez important au début du stage, incluant la lecture d'articles, la compréhension de modèles existants et la réalisation de plusieurs tests sur différents petits graphes de désassemblage.

Ces expérimentations ont permis d'assimiler progressivement les concepts avant de passer à la phase de développement méthodologique du projet de manière adéquate.

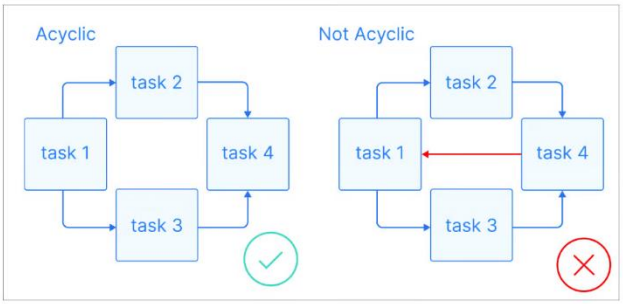
3.2. Modélisation du produit

La première étape du projet a consisté à modéliser le produit à désassembler sous une forme exploitable par les algorithmes de planification.
Le désassemblage d'un produit est un processus séquentiel soumis à des contraintes de précédence ou certaines pièces ne peuvent être retirées qu'après le démontage d'autres composants.

Afin de représenter ces relations logiques, le produit est modélisé sous la forme d'un graphe orienté acyclique (DAG), où :

- Les nœuds représentent les composants,
- Et les arêtes indiquent les dépendances entre eux.

Ce choix de modélisation permet d'identifier toutes les séquences de démontage possibles tout en respectant les contraintes structurelles du produit.
Elle constitue la base du Disassembly Sequence Planning (DSP) et sert de support à la génération et à la replanification des séquences.



"Figure 5 : *Explication visuelle des graphes acycliques*"

Chaque composant est décrit par un ensemble d'attributs simples tels que :

- Un identifiant unique,
- Un profit estimé (valeur récupérable),
- Un coût d'opération,
- Et un temps d'exécution associé.

Ces informations sont stockées sous un format JSON, un format simple qui peut donc être relu par les programmes du projet pour faire des tests, simuler des démontages ou visualiser les graphes.
Les dépendances entre composants y sont définies sous forme d'arêtes (A → B), indiquant que le démontage de *A* est nécessaire avant celui de *B* par exemple.

Dans le cadre de ce projet, plusieurs produits ont été modélisés :

- des produits générés artificiellement pour tester le comportement des algorithmes sur des graphes de tailles variables,
- des produits inspirés de cas réels issus de la littérature (par exemple un boîtier électronique ou un boitier de vitesse),
- ainsi que des produits venant de benchmarks reconnus (Scholl, Otto, ...)

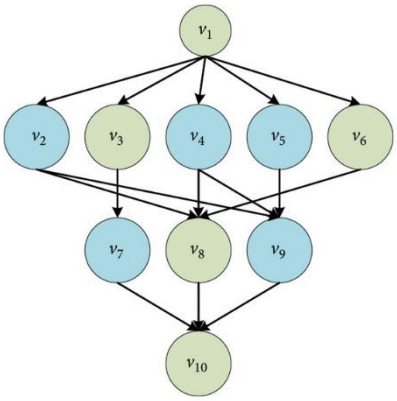
Ces différents graphes ont permis d'évaluer les performances et la robustesse des méthodes proposées dans des contextes de complexité croissante.

3.3. Planification initiale (DSP)

3.3.1 Formulation du problème

Le désassemblage séquentiel ici consiste à déterminer l'ordre optimal dans lequel les composants d'un produit doivent être retirés, tout en respectant leurs contraintes de dépendance.
Chaque produit est modélisé sous la forme d'un graphe orienté $\boldsymbol{G = (V, E)}$, où :

- $\boldsymbol{V}$ représente l'ensemble des composants,
- $\boldsymbol{E}$ l'ensemble des arcs définissant les relations de précédence : un arc (i, j) signifie que le composant *i* doit être retiré avant *j*.



"Figure 6 — Exemple de graphe de démontage classique G = (V,E)"

Le but du **Disassembly Sequence Planning (DSP)** est de trouver une séquence valide de démontage maximisant une certaine performance tout en respectant les contraintes physiques et logiques.
Dans le cadre du projet **LAsSy**, la planification vise deux cas :

- le **désassemblage complet**, où tous les composants sont retirés ;
- le **désassemblage sélectif**, le plus important car le plus fréquent en pratique, où seules certaines pièces cibles sont démontées pour valorisation ou maintenance.

Chaque composant $\boldsymbol{i \in V}$ est associé à plusieurs paramètres :

- $\mathbf{P_i}$: profit ou valeur récupérable,
- $\mathbf{C_i}$: coût de démontage,
- $\mathbf{T_i}$: durée estimée de l'opération.

L'objectif du DSP est donc de maximiser la valeur totale tout en minimisant les coûts et le temps d'exécution, selon une fonction multicritère choisie :

$$\textbf{Score global} = \boldsymbol{w_1 \sum P_i - w_2 \sum C_i - w_3 \sum T_i}$$

La somme des coefficients de pondération vérifie $\Sigma w_i = 1$.

où $\boldsymbol{w_1}$, $\boldsymbol{w_2}$ et $\boldsymbol{w_3}$ sont des coefficients de pondération permettant d'ajuster les priorités selon le scénario étudié (économique, énergétique, ou temporel).

Le problème est de nature **NP-difficile** ce qui veut dire que le nombre de séquences possibles croît de manière exponentielle avec le nombre de composants.
C'est pourquoi différentes approches tel que modèles exacts, heuristiques et métaheuristiques ont été étudiées pour proposer des solutions réalisables dans des temps de calcul acceptables.

3.3.2 Méthodes exactes : modèle MILP (baseline)

Avant de recourir aux approches heuristiques, une première étape a consisté à établir un modèle exact de référence basé sur un **programme linéaire mixte en nombres entiers (MILP).**
Ce modèle sert principalement de point de comparaison fiable pour évaluer la qualité des séquences générées par nos méthodes approchées.

Objectif concret du modèle

L'idée est de déterminer une séquence de démontage optimale maximisant le profit net de désassemblage, tout en respectant les contraintes de précédence déjà décrites dans la section précédente.
Contrairement aux approches heuristiques, le MILP garantit une solution exacte, mais au prix d'un temps de calcul très souvent élevé.

Voici ce que modèle cherche à maximiser :

$$Z = \sum_i (P_i - C_i) \cdot x_i$$

où $x_i \in \{0, 1\}$ indique si le composant *i* est démonté (1) ou non (0).
Les contraintes garantissent simplement la cohérence logique du démontage (ordre et cibles obligatoires).
Concernant T_i, il est uniquement utilisé à titre informatif pour l'analyse, sans impacter la fonction objective.

Implémentation

Le modèle a été implémenté en **Python** avec la bibliothèque **PuLP**, en utilisant les solveurs **CPLEX** ou **CBC**.
Ainsi nous avons un module pour **générer automatiquement le modèle MILP** à partir des graphes d'entrée, en intégrant les relations de précédence, les coûts et les durées associées à chaque composant.
Puis un second module permettant de **lancer la résolution**, **analyser les résultats** (séquence optimale, temps de calcul, valeur de l'objectif) et **les enregistrer** de manière structurée.
Cette automatisation a permis de **comparer facilement les performances** du modèle exact avec celles des approches heuristiques et métaheuristiques testées par la suite.

Limites et utilité prévue

Le MILP devrait donner des résultats exacts et cohérents pour de petits et moyen graphes, confirmant la validité des formulations et des dépendances.
Il a donc été utilisée principalement comme **référence de validation** pour évaluer la qualité des séquences obtenues par GRASP et les recherches locales dans la suite du projet.

3.3.3 Méthodes heuristiques et métaheuristiques étudiées

Après la modélisation exacte du problème, il a été nécessaire d'explorer des approches plus légères et flexibles, capables de produire maintenant des séquences valides en un temps raisonnable.
Comme on peut le constater plutôt, la complexité du DSP rend les méthodes exactes rapidement inapplicables dès que le nombre de composants devient beaucoup trop nombreux.
Donc les heuristiques et métaheuristiques offrent ainsi un bon compromis entre rapidité, qualité de solution pour la suite.

Étude bibliographique et sélection des approches

Les publications sur le sujet telles que (Guo et al., 2021 ; Ye et al., 2022 ; …) montrent que les méthodes hybrides combinant construction gloutonne, aléa contrôlé et recherche locale sont les plus efficaces pour ce type de problème.

Cette analyse nous a conduit à retenir un ensemble de méthodes représentatives :

- des **heuristiques simples** pour les premières validations du modèle,
- et plusieurs **métaheuristiques** plus puissantes pour la suite.

Heuristiques gloutonnes

Les heuristiques simples ont donc constitué la première étape expérimentale.
Leur principe est de sélectionner, à chaque itération, le composant le plus "intéressant" selon un **score local** combinant profit, coût et durée choisie.
Ces approches sont rapides et intuitives, mais purement locales : elles peuvent générer des séquences correctes mais sous-optimales.

Elles ont surtout servi à **comprendre et valider la cohérence des graphes et des dépendances**, et à vérifier que les critères économiques et temporels produisaient des comportements réalistes avant de passer à des méthodes plus avancées.

Introduction des métaheuristiques

Pour dépasser ces limites, nous avons ensuite étudié plusieurs métaheuristiques, qui sont des algorithmes d'optimisation capables d'explorer l'espace des solutions de manière plus globale.

Le principe général est de :

1. Générer une solution initiale (souvent issue d'une heuristique),
2. L'améliorer par une recherche locale,
3. Répéter le processus en contrôlant le niveau d'aléatoire.

Donc les algorithmes considérés ont été :

- **GRASP**
- **VND**
- **MNS**
- **Tabu Search**

Leur fonctionnement sera expliqué dans des parties spécifiques plus loin dans le rapport.

Choix du GRASP comme approche principale

Ce choix s'est imposé après comparaison théorique et expérimentale.
La littérature spécialisée (notamment Resende & Ribeiro, 2019) montre que cette méthode offre un excellent compromis entre vitesse d'exécution, qualité des solutions à des structures de graphe complexes.

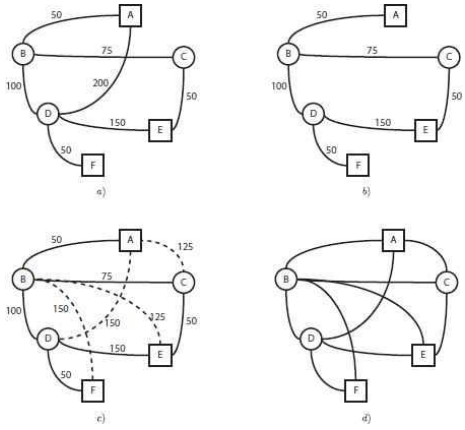
GRASP s'intègre naturellement dans une architecture assez modulaire et permet d'envisager facilement des extensions.
De ce fait dans le projet LAsSy, il a été retenu comme noyau principal de la planification, les autres méthodes (VND, MNS, Tabu) servant de modules d'amélioration complémentaires, détaillés par la suite.

3.3.4 GRASP : principe et adaptation au DSP

Principe général de GRASP

Le **GRASP (Greedy Randomized Adaptive Search Procedure)** est une méthode métaheuristique itérative combinant deux phases principales :

1. **Construction d'une solution initiale** de manière gloutonne mais contrôlée par un facteur d'aléatoire,
2. **Amélioration locale** de cette solution à l'aide d'une recherche locale.



"Figure 7 : Principe du GRASP sur un graphe de désassemblage."

(*a) Graphe initial, (b) construction d'une solution gloutonne, (c) amélioration locale, (d) séquence finale optimisée.*

Chaque itération produit une solution candidate, et la meilleure parmi toutes les itérations est conservée.
Cette approche permet de mixer exploration et exploitation, évitant le blocage dans des optima locaux tout en maintenant une bonne convergence.

Phase constructive

La phase constructive repose sur la création d'une **liste restreinte de candidats (RCL)**.

À chaque étape, les composants démontables sont évalués selon un **score** basé sur la fonction multicritère du projet vu précédemment :

Les meilleurs composants (selon un seuil α) sont placés dans la RCL.

L'un d'eux est ensuite sélectionné aléatoirement pour construire la séquence.

Le paramètre **α ∈ [0,1]** contrôle le degré d'aléatoire :

- α proche de 0 → choix très glouton (solutions de bonne qualité, mais peu diversifiées),
- α proche de 1 → choix plus aléatoire (exploration plus large).

Phase d'amélioration

Une fois une séquence complète construite, la phase d'amélioration locale est appliquée.
Elle consiste à explorer le voisinage de la solution actuelle par exemple en échangeant deux composants afin d'améliorer le score global si possible tout en respectant les contraintes de précédence.

Cette étape a été structurée pour accueillir différentes stratégies, dont celles étudiées plus loin (VND, MNS et Tabu Search), afin de renforcer la robustesse du GRASP face à des graphes complexes.

Adaptation au DSP et version réactive

Dans le projet LAsSy, le GRASP a été adapté pour traiter le **désassemblage sélectif** principalement, où seules certaines pièces cibles doivent être retirées.
Donc la fonction de score a été ajustée pour pondérer les composants cibles plus fortement, garantissant leur démontage prioritaire.

De plus, en extension une **version réactive du GRASP** a été implémentée.
Plutôt que de fixer α manuellement, le système ajuste automatiquement sa valeur selon la qualité moyenne des solutions obtenues lors des itérations précédentes.
Ce mécanisme permet de trouver un équilibre dynamique entre exploitation et exploration, ce qui améliorera la stabilité et la diversité des séquences.

Avantages prévus

Cette version réactive du GRASP présente plusieurs atouts :

- **Flexibilité** face à la diversité des graphes (taille, dépendances, cibles).
- **Qualité de solution** souvent proche de l'optimum du MILP.
- **Scalabilité** : temps de calcul quasi linéaire par rapport à la taille du graphe.

Elle constitue donc une partie importante de la planification initiale du système, avant d'être couplée à la phase de replanification adaptative du bloc suivant.

3.3.5 Recherche locale : VND, MNS, Tabu Search

Rôle et objectif

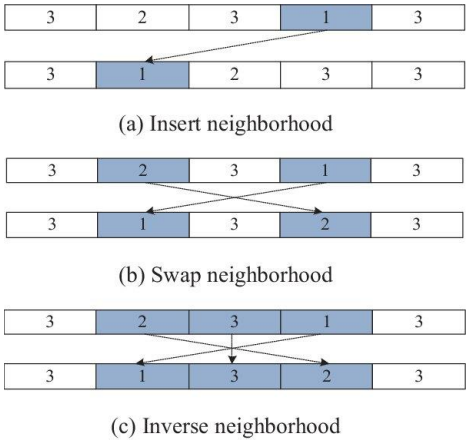
La recherche locale constitue la deuxième phase du processus de planification dites plutôt, après la génération initiale de séquences par le GRASP.
Elle vise à améliorer les solutions existantes en explorant des variations mineures (ou voisinages), afin de trouver une séquence de démontage plus performante.
Cette étape est essentielle pour affiner les résultats et éviter que l'algorithme ne reste bloqué dans des configurations sous-optimales.

Stratégies de voisinage

Plusieurs types de mouvements connus ont été étudiés, permettant de créer des solutions voisines à partir d'une séquence existante :

- **Insertion (insert)** : déplacement d'un composant à un autre emplacement valide,
- **Échange (swap)** : inversion de la position de deux composants,
- **Inversion (reverse)** : inversion d'un sous-ensemble ordonné si les dépendances le permettent.

Ces opérations, combinées à un contrôle de faisabilité topologique, garantissent que toutes les séquences générées restent physiquement réalisables.



"Figure 8 : Stratégies de voisinage."

Méthodes étudiées

Nos trois approches issues de la littérature testées pour explorer efficacement l'espace des solutions.

Variable Neighborhood Descent (VND)

Le **VND** applique successivement plusieurs types de voisinages.
Dès qu'une meilleure solution est trouvée, la recherche reprend depuis ce nouveau point, ce qui permet d'explorer plus largement tout en assurant une convergence stable.

Multi-Neighborhood Search (MNS)

Le **MNS** reprend le principe du VND mais choisit le type de voisinage de manière dynamique selon les performances récentes.
Cette méthode favorise l'adaptation au type de graphe (dense, hiérarchique ou sélectif).

Tabu Search

La **Tabu** intègre une mémoire temporaire pour interdire le retour à certaines solutions déjà visitées.
Ce mécanisme de "tabou" empêche les cycles et encourage l'exploration de nouvelles zones de l'espace des solutions.
Cette méthode peut accepter ponctuellement des solutions légèrement moins bonnes pour éviter les optima locaux, ce qui justifie l'usage d'une mémoire tabou.
Bien que plus coûteuse en calcul, elle se révèle utile pour tester la stabilité du système et inspirer la future logique de replanification adaptative.

Choix de la stratégie finale

Après expérimentation, la combinaison **GRASP + VND** a été retenue comme la plus équilibrée.
Le VND offre une amélioration significative du score moyen sans accroître excessivement le temps de calcul, contrairement à la recherche tabou ou au MNS plus exigeants.
De plus, le VND s'intègre naturellement dans la structure du GRASP réactif, car il réutilise les voisinages déjà explorés durant la phase de construction.

Les deux autres approches demeurent toutefois pertinentes pour certains cas complexes :

- **MNS** pour renforcer la diversification lorsque le graphe est très ramifié,
- **Tabu Search** pour garantir la stabilité sur de grands ensembles de dépendances.

Paramètres et critères d'arrêt

Les paramètres communs aux méthodes locales sont :

- **Nombre maximal d'itérations locales** : limite le temps d'exploration,
- **Seuil de stagnation** : arrêt si aucune amélioration n'est observée après N itérations,
- **Temps d'exécution total** fixé pour garantir la reproductibilité des tests.

Ces critères assurent un comportement prévisible et évitent les suroptimisations coûteuses.
Les réglages précis (valeurs de N, voisinages testés, tolérances) sont décrits plus tard.

Synthèse

L'ajout d'une recherche locale a permis d'améliorer sensiblement la qualité des séquences obtenues après le GRASP réactif.
Le VND s'est imposé comme une méthode simple, rapide et efficace, renforçant la robustesse de la planification initiale.

3.3.6 Validation et cohérence des séquences

Objectif de la validation

La phase de validation vise à s'assurer que les séquences générées par les différentes approches (MILP, GRASP et variantes locales) sont cohérentes et réalisables d'un point de vue logique et topologique.
Cette étape ne cherche pas à comparer les performances des méthodes, mais à garantir que les solutions produites respectent strictement la structure du graphe et les contraintes de précédence définies lors de la modélisation.

Vérification de la cohérence topologique

Chaque séquence est vérifiée afin de confirmer la validité de l'ordre de démontage.
Un module dédié, réalisé sur un parcours de graphe inspiré de Dijkstra, a été fait pour :

- contrôler la conformité des relations de précédence entre composants,
- détecter des incohérences (cycles, ruptures de dépendance),
- assurer la complétude du parcours jusqu'à la dernière pièce ciblée démontée.

L'algorithme valide donc automatiquement chaque séquence avant son évaluation, garantissant une base de données de résultats fiable.

Vérification logique et structurelle

Outre la structure du graphe, une seconde validation porte sur la logique d'exécution :

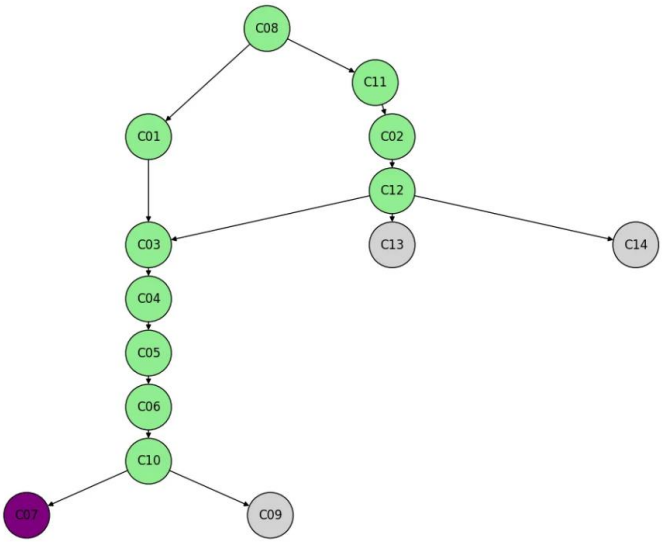
- chaque composant ne doit apparaître qu'une seule fois dans la séquence,
- les pièces cibles et leurs dépendances sont correctement incluses dans le cas d'un désassemblage sélectif,
- les séquences partielles ou invalides sont automatiquement écartées.

Ce contrôle a été implémenté dans un script commun à toutes les méthodes, garantissant une évaluation homogène et indépendante.

Visualisation et contrôle manuel

Enfin pour compléter les vérifications automatiques, de la visualisation de graphe a été mis aussi en place.
Il représente les séquences validées sous forme de graphes colorés :

- les nœuds correspondent aux composants,
- les arcs représentent les contraintes de précédence,
- et les couleurs indiquent l'état du démontage (non traité, en cours, terminé).



"Figure 9 : Exemple d'une séquence validée."

Vert (composant démonté avec succès), Gris (non traité), Violet (cible)

Cette visualisation a permis de confirmer la validité structurelle des séquences et d'identifier rapidement les éventuelles anomalies lors des tests si besoin.

Synthèse

La validation constitue une étape clé du processus méthodologique.
Elle garantit que toutes les séquences générées qu'elles proviennent du modèle exact ou des approches heuristiques sont structurellement correctes et exploitables.
Ce contrôle préalable assure la fiabilité des données utilisées pour les expérimentations de la prochaine phase : la replanification adaptative.

3.4. Replanification adaptative

La planification initiale fournit une séquence valide en conditions nominales.

En pratique, le désassemblage rencontre souvent des aléas (pièce bloquée, vis foirée, outil inadapté, mauvaise orientation, casse, …) qui rendent la séquence courante inapplicable.
Notre bloc adaptatif ajoute donc une sorte de réflexe qui va :

1. **Détecter** un échec à partir d'observables (capteurs, timeouts, retours d'API robot/vision, logs, humain).
2. **Décider** de l'action la plus pertinente (contourner, changer d'outil, retirer un obstacle, replanifier localement, …).
3. **Agir** et **resynchroniser** le plan avec l'état réel du produit (mise à jour du graphe, reprise au bon endroit).

L'objectif n'est pas de remplacer la planification, mais de la rendre préparée aux imprévus avec des boucles d'adaptation courtes.

3.4.1 Détection des échecs

Principes généraux

La détection regroupe plusieurs règles simples et testables à partir de signaux disponibles pendant l'exécution.

On distingue deux familles de symptômes :

Signaux physiques / capteurs :

- **surconsommation** ou **pic de couple/force** → suspecte un blocage mécanique ;
- **glissement / absence de prise** (écarts position) → mauvaise préhension ;
- **échecs de vision** répétés → mauvaise localisation / orientation ;
- **compteur d'essais** dépassé → instabilité sur l'opération.

Signaux logiciels / temporels :

- **timeout** sur une action → opération non réalisée dans la fenêtre attendue ;
- **retour d'erreur** → cobot / caméra / humain ;
- **incohérence logique** (état du composant ≠ attendu) → divergence plan / réalité.

Ces règles sont **paramétrées** par des seuils modifiables (ex. force max, nombre d'essais, durée max), centralisés pour rester faciles à ajuster si besoin.

Événements de détection

Chaque anomalie est encapsulée dans une **structure d'événement** qui sert d'entrée à la décision.
Contenu minimal et lisible (structuré pour une analyse claire) :

- **type** (blocage, absence de prise, casse suspectée, outil inadéquat, échec vision, timeout, etc.) ;
- **cible** (id composant / opération) ;
- **horodatage** et **compteur d'essais** ;
- **métriques clés** (valeur de couple/force mesurée, temps écoulé, code d'erreur) ;

Cette structuration permet de **loguer** proprement et d'alimenter la **politique de décision** de façon déterministe aujourd'hui… et pour la **logique floue** demain (module fuzzy).

Règles typiques choisies

- **Blocage mécanique** : couple > seuil et aucune progression → *event = "blocked"*
- **Mauvaise prise** : force trop faible + glissement + N essais → *event = "no_grip"*
- **Orientation incertaine** : vision échoue N fois → *event = "pose_fail"*
- **Outil inadapté** : échec couple + empreinte non reconnue → *event = "wrong_tool"*
- **Timeout opération** : durée > fenêtre max → *event = "timeout"*

Chaque règle reste indépendante et peut être testée seule.

Les seuils sont spécifiques à l'outil/opération.

Tests unitaires et log

Des **tests unitaires** valident les scénarios courants.
Les logs enregistrent : l'événement synthétique, les métriques captées et la décision prise ensuite.
Cela servent à **rejouer** des cas, **affiner les seuils**, et **documenter** les choix durant l'analyse futur.

3.4.2 Prise de décision adaptative

Principe général

Une fois un échec détecté, le système doit **décider quelle action entreprendre** pour continuer le désassemblage sans interrompre totalement la séquence.
En effet, la prise de décision repose actuellement sur une **logique déterministe**, structurée autour de conditions if/else.
Chaque type d'événement déclenche une **politique de réaction locale**, sélectionnée selon la **gravité**, le **type d'opération** et la **possibilité de contournement**.

Cette logique simple garantit une **réactivité immédiate**, une **traçabilité claire**, facilite nos tests et l'intégration ultérieure de mécanismes plus évolués prévues.

Structure de la politique de décision

Le cœur de cette partie est la fonction *choose_action()*.
Elle prend en entrée un événement de détection et retourne l'action la plus appropriée parmi les quatre possibles (Bypass, Change Tool, Destroy ou Replan) en fonction du type et du contexte de l'échec détecté.

Cette approche garantit que le système réagit toujours, même en cas d'événement inconnu (via un fallback par défaut : replan).
Chaque décision est ensuite enregistrée pour avoir une traçabilité dans les logs d'exécution.

Hiérarchie et priorisation

Les actions suivent un **ordre de priorité** cohérent avec les principes du désassemblage :

1. **Préserver la séquence** (bypass) si possible.
2. **Adapter l'outil ou la méthode** avant d'abandonner la pièce.
3. **Utiliser la destruction** uniquement en dernier recours.
4. **Replanifier** si la structure logique du graphe est affectée.

Cette hiérarchie traduit donc une politique de minimisation des pertes et de maintien du flux opérationnel autrement dit, notre système cherche à conserver la continuité avant d'envisager une réorganisation complète.

Préparation pour le module fuzzy

La structure du code et du pipeline a été pensée dès le départ pour accueillir un **système de logique flou** (fuzzy logic).
Celui-ci remplacera progressivement les conditions binaires (if/else) utilisé juste pour nos tests.

Chaque événement pourra être évalué par plusieurs **degrés d'appartenance** (faible, moyen, élevé) sur différents critères (force, temps, confiance, coût, ...), produisant une décision plus **souple et graduelle**.

Cette intégration se fera sans modifier la structure globale, grâce à notre modularité actuelle.

Validation et tests

La fonction de décision a été testée dans des scénarios unitaires et combinés, reproduisant des échecs fréquents choisies au préalable cité précédemment.

Puis nos tests vérifient : que chaque type d'événement produit une action cohérente,

que la priorisation est respectée,

et qu'aucune situation ne mène à un état indéfini.

3.4.3 Exécution des actions adaptatives

Une fois la décision prise, le système doit mettre en œuvre l'action correspondante et actualiser la séquence de désassemblage.
Cette étape relie donc la théorie (règle choisie) à la pratique (mise à jour du plan et poursuite de l'exécution).
L'objectif est de garantir la continuité du processus, sans perte de cohérence dans le graphe ni réinitialisation complète du scénario.

Chaque action adaptative agit à deux niveaux :

- **opérationnel**, en modifiant localement l'exécution (outil, trajectoire, stratégie) ;
- **logique**, en mettant à jour le graphe ou la séquence active pour refléter le nouvel état du produit.

Structure du module d'actions

Les actions sont définies sous forme de **fonctions indépendantes** dans le module `actions.py`.
Elles partagent une interface commune :

```
def execute_action(action_type, context):
    ...
    return updates_action
```

"Figure 10 : Exemple d'interface utilisée."

où *context* contient :

- le graphe courant de désassemblage ;
- le nœud ou composant concerné ;
- l'état du robot et la liste des actions déjà effectuées ;
- un log pour assurer la traçabilité.

Cette conception assez modulaire permet de tester chaque action séparément et de les combiner dans des scénarios plus complexes (ex. séquences avec multiples échecs).

Les quatre types d'actions

a) Bypass

L'action *bypass* consiste à **contourner le composant en échec si une alternative valide** dans le graphe est possible.
Cette action est prioritaire lorsque le composant bloqué n'est pas critique à la progression globale du démontage.
Le module met à jour le graphe en supprimant le nœud concerné et en reconnectant ses dépendances directes.
Cette stratégie est efficace pour contourner des pièces non essentielles ou des sous-assemblages secondaires tout en gardant la séquence de base.

b) Change Tool

L'action *change_tool* simule un **changement d'outil** avant de relancer l'opération échouée.
Ce mécanisme ne modifie pas le graphe mais ajoute une **nouvelle tentative** dans les logs, avec un paramètre d'outil différent (ex. tournevis électrique → manuel).
Cette action reflète un comportement humain réaliste et permet d'éviter un arrêt prématuré du processus.

c) Destroy

Dans certains cas assez urgents, la seule manière d'extraire un composant est de recourir à un **démontage destructif**.
L'action *destroy* est alors déclenchée, ce qui met à jour :

- le graphe (remplacement du nœud par un état "détruit") ;
- la fonction de score (réduction du profit associé) ;
- et les logs d'événements.

Cette stratégie montre un compromis entre **productivité** et **préservation de la valeur récupérable**.

d) Replan

L'action *replan* est la plus complexe : elle **reconstruit localement la séquence de démontage** à partir de l'état actuel du graphe.
Le système supprime les éléments déjà retirés, puis relance le module de planification sur le graphe restant.
Cette approche permet de retrouver une cohérence globale tout en limitant la replanification à une zone affectée par l'échec, c'est l'une des actions les plus utiles si aucune des actions son possible.

Interaction entre actions et graphe

Toutes les actions s'exécutent en lien direct avec la structure de graphe utilisée dans la planification initiale.
Les fonctions du module *overlay.py* assurent :

- la **mise à jour des dépendances** après suppression ou modification d'un nœud ;
- la **vérification de la validité topologique** (aucune contrainte brisée) ;
- la **synchronisation des états** entre simulation et exécution réelle.

Cela garantit que chaque décision adaptative reste **cohérente** avec les contraintes initiales, évitant ainsi les séquences impossibles ou contradictoires.

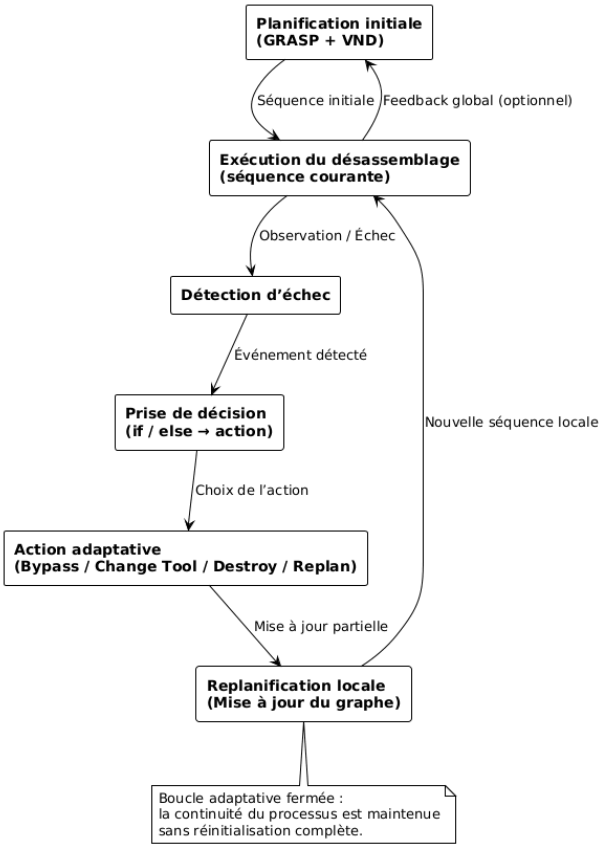
3.4.4 Pipeline Adaptatif et conclusion du bloc

Cet ensemble du module adaptatif vient s'intégrer au pipeline global du projet LAsSy, reliant la planification initiale et la replanification locale en cas d'aléa.
Ce pipeline assure la coordination entre les différentes étapes développées précédemment (détection, décision, action) et garantit la continuité du désassemblage face aux imprévus.

Il se compose d'une boucle unique reliant la planification, l'exécution et la replanification, assurant ainsi une bonne adaptation du système sans intervention humaine.
Chaque module communique avec les autres par échange d'états et mise à jour du graphe, ce qui permet de maintenir la cohérence de l'ensemble.

Par ailleurs, ce fonctionnement boucle la méthodologie du projet : la partie planification (GRASP + VND) qui fournit les séquences initiales, tandis que le module adaptatif permet leur ajustement dynamique comme prévu.

Nous obtenons enfin une architecture complète, modulaire qui est prête à accueillir à terme la suite du projet futur.



"Figure 11 : Pipeline adaptatif du projet."

4. Résultats et Analyses

4.1 Présentation des expérimentations

Pour évaluer la performance et la stabilité du pipeline développé, nous avons mené une large campagne d'expérimentations sur **plus de 246 instances** de désassemblage, représentant des graphes de **10 à 1000 nœuds**.
Ces instances proviennent de trois sources :

- des graphes artificiels générés selon les contraintes du projet (produits réalistes avec dépendances logiques et cibles définies),
- des instances issues de la littérature récente,
- et des benchmarks publics tels que **Scholl** et **Otto**, couramment utilisés pour l'évaluation des algorithmes de planification.

Chaque produit est converti dans un format JSON compatible avec les scripts de test.
Les algorithmes MILP, GRASP (et ses variantes locales VND, MNS, Tabu) ainsi que Dijkstra sont ensuite exécutés automatiquement via un script Bash préparé à cet effet.
Ce script gère la séquence complète des tests, l'enregistrement des résultats et la sauvegarde des logs.

Chaque exécution produit un dossier dédié contenant :

- les séquences de démontage générées,
- les logs complets du solveur CPLEX,
- un tableau comparatif (temps, profit, gap),
- et, si nécessaire, une visualisation du graphe.

Cette organisation assure la traçabilité complète de toutes les expériences et permet de reproduire les résultats sans difficulté.

4.2 Etude de cas représentatif

L'instance **dagtest** sera prise comme exemple pour illustrer les démarches expérimentales.
Ce graphe de taille moyenne représente un désassemblage sélectif comportant une cible principale à atteindre.

Sur ce cas, MILP et GRASP + VND aboutissent à la **même séquence optimale.**
Le **GRASP** trouve cette solution en moins de **0,3 s**, contre **0,55 s** pour le **MILP**, ce qui démontre son efficacité.
L'algorithme **Dijkstra** a servi uniquement à vérifier la validité topologique du graphe, confirmant la cohérence de la séquence obtenue.

Ces résultats se généralisent à l'ensemble des petites et moyennes instances (moins de 300 nœuds), où le **GRASP + VND** maintient un **écart inférieur à 3 %** par rapport au MILP tout en réduisant fortement le temps de calcul.

Méthode	Profit total	Makespan	Temps (s)	Écart MILP	Statut
MILP (baseline)	333.0	22.0	0.55	0 %	Optimal
GRASP + VND	333.0	22.0	0.02	0 %	Completed
GRASP + Tabu	333.0	22.0	0.31	0 %	Completed
GRASP + MNS	333.0	22.0	0.03	0 %	Completed
Dijkstra (validation)	91.0	6.0	6.00	72.7 %	Completed

"Figure 12 : Résultats comparatifs sur l'instance dagtest."

Les détails complets des logs (CPLEX, CSV, terminal, …) sont fournis en annexe pour référence.

4.3 Résultats globaux et test de robustesse

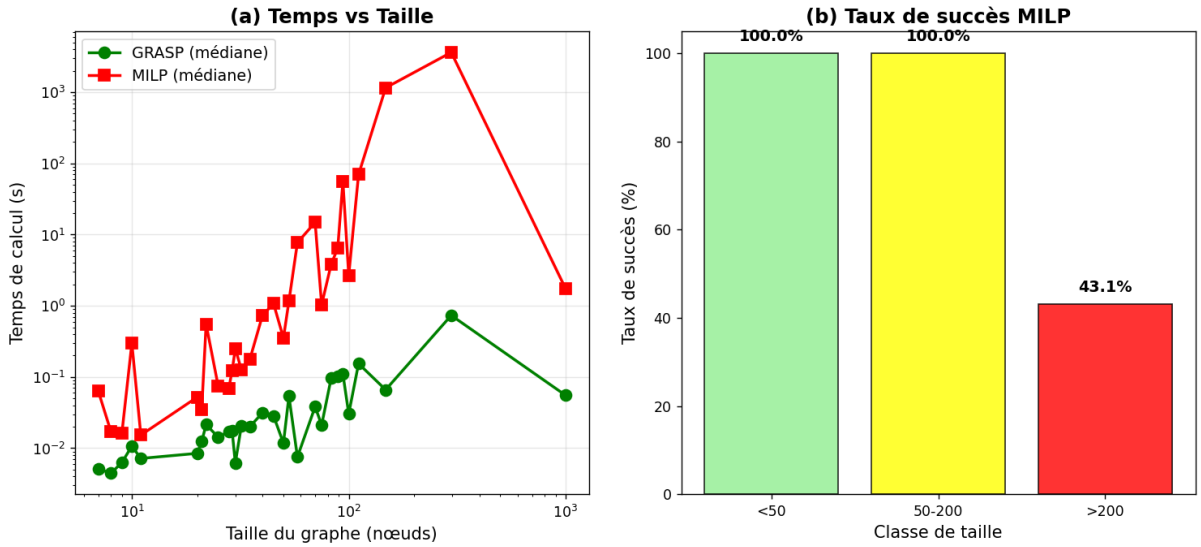
Pour évaluer la scalabilité et la robustesse du système, chaque instance a été testée **30 fois**, avec une **limite de 30 minutes par exécution**, afin de mesurer la stabilité du GRASP et de réduire l'impact du caractère aléatoire de la méthode.
L'ensemble des tests a duré environ **27 heures d'exécution continue**.

Chaque dossier de test contenait les résultats MILP, GRASP et Dijkstra, les logs complets et les tableaux comparatifs.
Cette structure a facilité le suivi des performances sur la durée et la comparaison automatique des méthodes.

Cependant, sur les très grandes instances (≥ 600 nœuds), le solveur CPLEX a rencontré des **erreurs mémoire** récurrentes dues à la taille du modèle MILP.
Malgré plusieurs ajustements (warm start, fichiers swap, réduction des variables), les calculs restaient instables.
Ces observations rejoignent celles de la littérature : le MILP devient rapidement inapplicable à grande échelle.

En revanche, le GRASP réactif couplé à une recherche locale a démontré une **excellente stabilité**.
Sur les instances où le MILP restait solvable, le **gap moyen** entre les deux approches est toujours inférieur à **3 %**, tandis que le temps de calcul reste quasi constant au-delà de 500 nœuds.
Ces résultats confirment la **scalabilité** et la **pertinence** du GRASP pour des applications industrielles de désassemblage automatisé.

.



"Figure 13 : Scalabilité et robustesse (résumé)"

4.4 Expérimentation du scheduler adaptif

Enfin, une série de tests a été consacrée au scheduler adaptatif, chargé de gérer les aléas pendant le désassemblage.
L'objectif était toujours de vérifier que le système **détecte**, **analyse** et **corrige** correctement les échecs sans rompre la cohérence du plan de base.

L'instance **dagtest** a une nouvelle fois été utilisée à titre d'exemple ici.

Les autres instances ont été traitées de manière similaire pour valider la généralité des comportements observés.

Chaque composant peut adopter trois **états** :

- **done** : démonté avec succès,
- **locked** : momentanément bloqué,
- **destroyed** : retiré de manière destructive.

Trois types d'événements d'échec choisis volontairement ont été simulés :

- **ACCESS_BLOCKED :** simulant un blocage d'accès à un composant,
- **TIMEOUT :** représentant un dépassement du temps alloué à l'opération,
- **BREAKAGE :** correspondant à la casse d'un composant au cours du démontage.

Les **logs détaillés** montrent que le système a réagi comme prévu :

- Lors du premier incident, la stratégie **bypass** a été choisie : le composant bloqué a été temporairement ignoré, et la séquence a continué sur un chemin alternatif valide.
- Puis du second événement, la politique **replan** a été appliquée : le scheduler a régénéré localement une sous-séquence à partir de l'état courant du graphe sans relancer toute la planification.
- Enfin, lors de la casse de pièce, l'action **destroy** a été déclenchée : le composant a été marqué comme irrécupérable avant la reprise du processus et la séquence a continué le chemin de base.

Malgré ses plusieurs échecs successifs, la **cible** a été atteinte dans tous les cas sans incohérence topologique.
Donc le pipeline complet s'est montré stable et entièrement traçable.
Ces observations, confirmées sur d'autres instances du jeu de tests, mettent en évidence la **stabilité**, la **traçabilité** et la **robustesse** du pipeline global face aux imprévus.

5. Conclusion Générale

5.1 Conclusion d'un point de vue projet

Ce projet m'a permis de développer une approche complète pour la planification et la replanification adaptative du désassemblage automatisé dans le cadre du **projet LAsSy**. L'objectif principal était donc de concevoir un système capable de générer des séquences de démontage valides, tout en s'adaptant aux imprévus rencontrés durant l'exécution.

État des lieux du projet :

- **Développement et validation du pipeline complet :** Nous avons pu réaliser une architecture intégrant la modélisation des produits, la planification initiale et un module adaptatif de replanification locale.
- **Planification et optimisation :** Le GRASP réactif, combiné à la recherche locale, permet de produire des séquences proches de l'optimum tout en réduisant fortement les temps de calcul.
- **Gestion des aléas :** Le module de détection–décision–action a permis de réagir automatiquement aux échecs (timeout, etc.) et de maintenir la continuité du processus.
- **Résultats :** Les tests ont confirmé la robustesse du système et la cohérence des séquences, validant donc le pipeline sur plusieurs graphes de désassemblage.

Difficultés rencontrées et solutions envisagées :

- **Complexité du DSP :** Le problème étant de nature NP-difficile, il a fallu trouver un équilibre entre qualité de solution et temps de calcul, en combinant heuristiques et métaheuristiques.
- **Modularité du code :** La gestion simultanée de la planification et de la replanification a nécessité une structuration claire du code et des interfaces entre les blocs.
- **Limites matérielles :** Les calculs MILP et certaines simulations de graphes complexes ont été contraints par les ressources disponibles (mémoire, processeur), ce qui a nécessité d'optimiser les scripts et de limiter la taille des instances traitées simultanément.

Suites et perspectives du projet :

- **Implémentation du module fuzzy :** Ajouter un raisonnement flou pour rendre la prise de décision plus souple et pondérée.
- **Rédaction d'un article scientifique :** Rédiger un article basé sur les résultats obtenus pour une conférence, workshop futur.
- **Validation sur cellule réelle :** Tester le pipeline sur la plateforme cobotique 4.0 du CERI SN pour valider le système en conditions réelles.
- **Extension multi-produit :** Adapter le modèle à différents produits industriels afin d'évaluer sa généricité et son intégration dans une chaîne de désassemblage complète.

Ce stage a par conséquent contribué à poser les bases d'un cadre d'optimisation hybride associant planification, adaptation et intelligence décisionnelle, ouvrant la voie à des travaux plus larges dans le domaine de l'industrie 4.0.

5.2 Conclusion d'un point de vue personnel

Mon stage au sein du centre de recherche **IMT Nord Europe (CERI SN)** m'a permis de renforcer mes compétences dans les domaines de la robotique, de l'intelligence artificielle et de l'optimisation, tout en participant à un projet à valeur scientifique et industrielle.

J'ai eu l'opportunité de travailler dans un environnement de recherche stimulant, où rigueur, autonomie et créativité étaient au cœur du processus.

Apports sur l'expérience professionnelle :

- **Compétences techniques :** J'ai acquis une solide expérience en algorithmique d'optimisation, en modélisation par graphes, et en conception de pipelines modulaires.
- **Gestion de projet :** La planification des livrables, les validations successives et les réunions hebdomadaires m'ont permis d'améliorer mon organisation et mon sens de la priorisation.
- **Travail en équipe et communication :** Les échanges réguliers avec les encadrants et chercheurs m'ont appris à présenter clairement mes résultats, à argumenter mes choix et à adapter mon discours à différents profils.

Savoir-faire et savoir-être :

- **Autonomie et initiative :** Le cadre semi-recherche du stage m'a poussé à trouver moi-même des solutions aux problématiques rencontrées, à expérimenter, et à documenter mes démarches.
- **Rigueur et adaptabilité :** Travailler sur un projet mêlant algorithmique, robotique et recherche m'a appris à rester méthodique, tout en m'adaptant aux contraintes d'un projet collaboratif évolutif.

Expérience personnelle et perspectives :

Cette immersion dans un environnement de recherche appliquée m'a conforté dans mon choix de carrière : je souhaite poursuivre dans la robotique intelligente et les systèmes autonomes, en combinant optimisation, IA et prise de décision.
Ce stage a également renforcé mon intérêt pour les projets collaboratifs européens et la valorisation de la recherche dans des applications concrètes.
En somme, cette expérience a été à la fois technique, scientifique et humaine.
Elle m'a permis de progresser en tant qu'ingénieur, de gagner en maturité professionnelle et de confirmer mon envie d'évoluer dans le domaine de la robotique et de l'intelligence artificielle appliquée à l'industrie.

6. Bibliographie

Désassemblage et planification (DSP)

- Guo, Y., Liu, G., & Tian, G. (2021). *Disassembly sequence planning: A survey*. *Journal of Manufacturing Systems*, 60, 736–751.
- Tian, G., Liao, W., & Lee, J. (2019). *Multi-objective disassembly sequence planning in uncertain environments*. *Robotics and Computer-Integrated Manufacturing*, 57, 172–184.
- Xu, W., Liao, H., & Lee, C. (2020). *A self-evolving system for robotic disassembly sequence planning under uncertainty*. *Procedia CIRP*, 93, 1103–1108.
- Liu, Z., Xu, W., & Liao, H. (2022). *Selective disassembly sequence planning under uncertainty using improved heuristics*. *Journal of Cleaner Production*, 356, 131–146.

Métaheuristiques (GRASP, VND, Tabu Search)

- Feo, T. A., & Resende, M. G. C. (1995). *Greedy randomized adaptive search procedures*. *Journal of Global Optimization*, 6(2), 109–133.
- Resende, M. G. C., & Ribeiro, C. C. (2019). *Greedy randomized adaptive search procedures: Advances and extensions*. In *Handbook of Metaheuristics* (3rd ed.). Springer.
- Gendreau, M., & Potvin, J.-Y. (Eds.). (2010). *Handbook of Metaheuristics* (2nd ed.). Springer.

Méthodes exactes et optimisation linéaire (MILP)

- Scholl, A., & Becker, C. (2006). *Exact and heuristic procedures for assembly line balancing*. *European Journal of Operational Research*, 168(3), 666–693.
- Battaïa, O., & Dolgui, A. (2013). *A taxonomy of line balancing problems and solution approaches*. *Int. J. of Production Economics*, 142(2), 259–277.
- Pinedo, M. (2016). *Scheduling: Theory, algorithms, and systems* (5th ed.). Springer.

Ordonnancement adaptatif et replanification

- Liao, H., Xu, W., & Lee, C. (2022). *Adaptive scheduling in dynamic disassembly systems based on fuzzy decision models*. *IEEE Transactions on Automation Science and Engineering*, 19(4), 4000–4012.
- Zhao, Y., & Tang, Q. (2014). *Fuzzy reasoning Petri nets for disassembly decision-making under uncertainty*. *Expert Systems with Applications*, 41(16), 7148–7160.
- Battiti, R., & Brunato, M. (2014). *Reactive search optimization: Learning while optimizing*. Springer.

7. Annexes

Environnement de travail

J'ai principalement travaillé sur mon ordinateur personnel pour le développement, la rédaction du code et la visualisation des résultats.
Les phases de test et d'exécution prolongée ont quant à elles été réalisées sur un pc plus puissant mis à disposition par l'IMT.

Pc de l'IMT :

- Dell Precision 3591
- Processeur : Intel Core Ultra 7 155H (16 cœurs)
- Mémoire RAM : 16 Go
- Carte graphique : Intel Graphics
- Système d'exploitation : Ubuntu 22.04 LTS (64 bits)

Ce pc, plus performant, a été utilisé pour faire tourner les builds complètes pendant plusieurs heures et tester la robustesse du pipeline.

Langage de programmation

L'ensemble du projet a été développé en Python.

Bibliothèques utilisées

Le projet repose sur plusieurs bibliothèques Python permettant à la fois la modélisation, le calcul et la visualisation :

- NumPy, Pandas : traitement et manipulation des données.
- NetworkX : création et gestion des graphes de désassemblage.
- Matplotlib, Seaborn : visualisation des résultats et séquences.
- SciPy : fonctions d'optimisation et calculs scientifiques.
- PuLP : modélisation et résolution du problème MILP (avec CPLEX ou CBC).
- IPython, Jupyter : environnement pour le prototypage et la visualisation.
- Pytest : tests unitaires et validation du code.

Structure du projet

Le projet est organisé de manière assez modulaire :

- **src/adaptive/** et **experiments2/** : cœur du scheduler adaptatif (détection, actions, décision).
- **scripts_tools/** et **experiments/** : méthodes DSP, GRASP, MILP et outils d'analyse.
- **data/** : instances de désassemblage et scénarios d'échecs.
- **results/** : résultats des benchmarks et logs d'exécution.
- **run/** : scripts d'exécution (batchs, monitor, demos, stress_tests).
- **tests/** : visualisation et validation du pipeline.

Cette structure sépare clairement la planification DSP et la partie adaptative, tout en assurant la traçabilité des résultats.

Pipeline et script clé

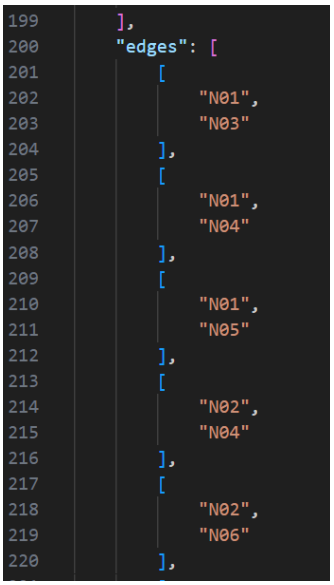
Le pipeline complet s'exécute à partir de *run/run_all_instances.py*, qui pilote l'ensemble du processus :

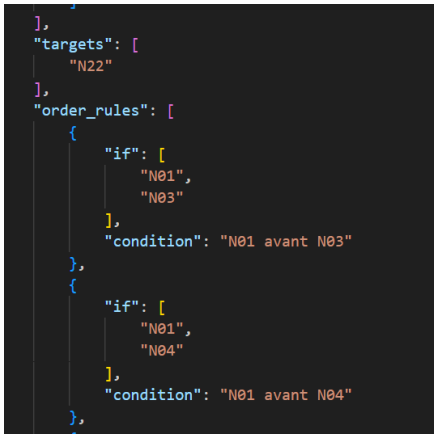
- Chargement de l'instance.
- Planification initiale.
- Simulation d'exécution et détection d'échecs.
- Replanification adaptative (si nécessaire).

Annexes finales : Partie expérimentale

L'instance **dagtest** a servi d'étude de cas principale tout au long du projet.
Cette instance a permis de tester la cohérence du pipeline complet de la planification initiale jusqu'à la replanification adaptative.

Extrait de l'instances JSON/TXT :





```
# Components (V)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

# Precedence arcs (E)
1 3
1 4
1 5
2 4
2 6
3 7
3 8
4 8
4 9
5 9
5 10
6 10
6 11
7 12
7 13
8 13
8 14
9 14
```

```
# Target components (T)
22

# Revenues (r)
1 20
2 18
3 22
4 19
5 21
6 17
7 23
8 20
9 18
10 22
11 19
```

```
# Costs (c)
1 5
2 4
3 6
4 5
5 4
6 3
7 7
8 5
9 4
10 6
11 5
```

```
# Durations (p)
1 100
2 90
3 110
4 95
5 105
6 80
7 120
8 100
9 90
10 110
```

```
orderrules
1 3 n01 avant n03
1 4 n01 avant n04
1 5 n01 avant n05
2 4 n02 avant n04
2 6 n02 avant n06
3 7 n03 avant n07
3 8 n03 avant n08
4 8 n04 avant n08
4 9 n04 avant n09
```

Voici des extraits des résultats bruts :

Fichier CSV :

results > compare > dagtest > compare_dagtest_20250925_165350.csv > data

```
instance,algorithm,profit_closure,makespan_closure,score_dsp,time,status,gap,targets,
dagtest,MILP_full_profit,333.0,22.0,253.0,0.55179762840271,Optimal,,1,"['N02', 'N01',
dagtest,grasp_vnd,333.0,22.0,253.0,0.02141880989074707,Completed,0.00%,1,"['N02', 'N
dagtest,grasp_tabu,333.0,22.0,253.0,0.3073160648345947,Completed,0.00%,1,"['N02', 'N
dagtest,grasp_mns,333.0,22.0,253.0,0.012432575225830078,Completed,0.00%,1,"['N02', '
dagtest,dijkstra_closure_min,91.0,6.0,21.0,6.0,Completed,72.67%,1,"['N01', 'N05', 'N
```

Log terminal Python (GRASP, MILP, comparaisons) :

```
Instance: dagtest
Cibles: ['N22']
 Exécution GRASP sur C:\Users\Waele\OneDrive\Documents\Etude\Cours\se
[DIAG] grasp_vnd run 0: seq_brute=['N02', 'N06', 'N11', 'N01', 'N17',
[DIAG] grasp_vnd run 0: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 1: seq_brute=['N02', 'N01', 'N03', 'N06', 'N07',
[DIAG] grasp_vnd run 1: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 2: seq_brute=['N02', 'N06', 'N01', 'N04', 'N03',
[DIAG] grasp_vnd run 2: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 3: seq_brute=['N01', 'N05', 'N03', 'N07', 'N02',
[DIAG] grasp_vnd run 3: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 4: seq_brute=['N02', 'N06', 'N01', 'N11', 'N05',
[DIAG] grasp_vnd run 4: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 5: seq_brute=['N01', 'N02', 'N06', 'N11', 'N05',
[DIAG] grasp_vnd run 5: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 6: seq_brute=['N01', 'N03', 'N05', 'N07', 'N02',
[DIAG] grasp_vnd run 6: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 7: seq_brute=['N01', 'N03', 'N02', 'N05', 'N04',
[DIAG] grasp_vnd run 7: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 8: seq_brute=['N01', 'N02', 'N06', 'N11', 'N05',
[DIAG] grasp_vnd run 8: profit=333.0, makespan=22.0, seq_filtered=['N
[DIAG] grasp_vnd run 9: seq_brute=['N02', 'N06', 'N01', 'N11', 'N05',
[DIAG] grasp_vnd run 9: profit=333.0, makespan=22.0, seq_filtered=['N
```

```
=== DIAGNOSTIC DIVERSITÉ GRASP ===
[grasp_vnd] Séquence brute: ['N02', 'N06', 'N11', 'N01', 'N17', 'N05', 'N10', 'N16
[grasp_vnd] Séquence filtrée: ['N02', 'N06', 'N11', 'N01', 'N17', 'N05', 'N10', '
[grasp_vnd] Score DSP: 253.0
[grasp_tabu] Séquence brute: ['N02', 'N06', 'N01', 'N11', 'N05', 'N03', 'N17', 'N
[grasp_tabu] Séquence filtrée: ['N02', 'N06', 'N01', 'N11', 'N05', 'N03', 'N17',
[grasp_tabu] Score DSP: 253.0
[grasp_mns] Séquence brute: ['N02', 'N06', 'N11', 'N01', 'N04', 'N03', 'N17', 'N08
[grasp_mns] Séquence filtrée: ['N02', 'N06', 'N11', 'N01', 'N04', 'N03', 'N17', '
[grasp_mns] Score DSP: 253.0
[dijkstra_closure_min] Séquence brute: ['N01', 'N05', 'N10', 'N16', 'N20', 'N22']
[dijkstra_closure_min] Séquence filtrée: ['N01', 'N05', 'N10', 'N16', 'N20', 'N22'
[dijkstra_closure_min] Score DSP: 21.0
```

```
=== COMPARAISON GRASP vs MILP ===
instance           algorithm  profit_closure  makespan_closure  score_dsp     time    status    gap  targets
 dagtest     MILP_full_profit           333.0              22.0      253.0 0.551798   Optimal              1 [N02, N01, N05, N04, N03, N08, N09,
 dagtest           grasp_vnd           333.0              22.0      253.0 0.021419 Completed  0.00%       1 [N02, N06, N11, N01, N17, N05, N10,
 dagtest          grasp_tabu           333.0              22.0      253.0 0.307316 Completed  0.00%       1 [N02, N06, N01, N11, N05, N03, N17,
 dagtest           grasp_mns           333.0              22.0      253.0 0.012433 Completed  0.00%       1 [N02, N06, N11, N01, N04, N03, N17,
 dagtest dijkstra_closure_min            91.0               6.0       21.0 6.000000 Completed 72.67%       1

Résultats sauvegardés dans: results/compare\dagtest\compare_dagtest_20250925_165350.csv
```

Log solveur (CPLEX) :

```
New value for mixed integer optimality gap tolerance: 0.0001
Version identifier: 22.1.1.0 | 2022-11-27 | 9160aff4d
CPXPARAM_TimeLimit                               7200
Tried aggregator 1 time.
MIP Presolve eliminated 181 rows and 107 columns.
MIP Presolve modified 13 coefficients.
Reduced MIP has 1025 rows, 706 columns, and 14786 nonzeros.
Reduced MIP has 706 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.02 sec. (7.05 ticks)
Found incumbent of value 333.000000 after 0.05 sec. (16.52 ticks)
Probing fixed 160 vars, tightened 0 bounds.
Probing time = 0.02 sec. (6.25 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 234 rows and 160 columns.
MIP Presolve modified 921 coefficients.
Reduced MIP has 791 rows, 546 columns, and 9437 nonzeros.
Reduced MIP has 546 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (5.88 ticks)
Probing fixed 16 vars, tightened 0 bounds.
Probing changed sense of 4 constraints.
Probing time = 0.02 sec. (12.93 ticks)
Tried aggregator 1 time.
Detecting symmetries...
MIP Presolve eliminated 39 rows and 20 columns.
MIP Presolve modified 260 coefficients.
Reduced MIP has 752 rows, 526 columns, and 8786 nonzeros.
Reduced MIP has 526 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.02 sec. (5.85 ticks)
Probing fixed 6 vars, tightened 0 bounds.
Probing time = 0.02 sec. (11.54 ticks)
Clique table members: 6989.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.02 sec. (11.52 ticks)
```

```
        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                          333.0000      424.0000           27.33%
*     0+    0                          377.0000      424.0000           12.47%
      0     0      424.0000   110      377.0000      424.0000      239   12.47%
*     0+    0                          424.0000      424.0000            0.00%
      0     0        cutoff             424.0000      424.0000      239    0.00%
Elapsed time = 0.27 sec. (246.86 ticks, tree = 0.01 MB, solutions = 3)

Root node processing (before b&c):
  Real time             =    0.27 sec. (246.89 ticks)
Parallel b&c, 12 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.27 sec. (246.89 ticks)

Solution pool: 3 solutions saved.

MIP - Integer optimal solution:  Objective =  4.2400000000e+02
Solution time =    0.27 sec.  Iterations = 239  Nodes = 0
Deterministic time = 246.89 ticks  (928.16 ticks/sec)

MILP problem relaxed to LP with fixed integer variables using
incumbent solution.
Version identifier: 22.1.1.0 | 2022-11-27 | 9160aff4d
CPXPARAM_TimeLimit                               7200
Tried aggregator 1 time.
LP Presolve eliminated 1206 rows and 813 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec. (1.15 ticks)

Dual simplex - Optimal:  Objective =  4.2400000000e+02
Solution time =    0.00 sec.  Iterations = 0 (0)
Deterministic time = 1.82 ticks  (1822.14 ticks/sec)
```

Fichier JSON – Exemple de scénarios d'échecs réalisés :

data > adaptive > generated_failures > dagtest > {} failures_3_on_dagtest.json > {} 1 > {} context > {} comp_id

```
[
  {
    "symptom": "ACCESS_BLOCKED",
    "signals": {
      "access_blocked": 1
    },
    "context": {
      "op_type": "unscrew",
      "tool_id": "T_hex",
      "alt_paths": true,
      "comp_id": {
        "id": "N02",
        "profit": 18,
        "cost": 4,
        "duration": 90,
        "milp_index": 2
      }
    }
  },
  {
    "symptom": "TIMEOUT",
    "signals": {
      "timeout": 1
    },
    "context": {
      "op_type": "unscrew",
      "tool_id": "T_hex",
      "comp_id": {
        "id": "N20",
        "profit": 22,
        "cost": 6,
        "duration": 110,
        "milp_index": 24
      }
    }
```

Log stress test adaptatif :

results > stress_logs > dagtest > stress_log_failures_3_on_dagtest.txt

```
[TEST ADAPTATIF SÉLECTIF 3]----------------------------
[INFO] Plan GRASP+VND (profit total = 333.0) : ['N01', 'N02', 'N05', 'N06', 'N10', 'N03', 'N11', 'N07', 'N04', 'N09', 'N16'
[INFO] Targets d'intérêt pour ce test : ['N22']
[INFO] Étape 1 : N01 (normal)
[INFO] État overlay : locked=set(), destroyed=set(), done={'N01'}
[DEBUG] detect_failure (injection) pour N02 => FailureEvent(comp_id='N02', op_type='unscrew', tool_id='T_hex', symptom=<Sym
[INFO] Injection d'échec sur le composant : N02 (étape 2)
[INFO] FailureEvent détecté : {"symptom": "ACCESS_BLOCKED", "signals": {"access_blocked": 1}, "context": {"op_type": "unscr
[DEBUG] FailureEvent objet : FailureEvent(comp_id='N02', op_type='unscrew', tool_id='T_hex', symptom=<Symptom.ACCESS_BLOCKE
[Decision] action=bypass
[INFO] Résultat action : success=True, updates={'next_nodes': ['N03', 'N05']}, notes=Bypass possible.
[INFO] État overlay : locked=set(), destroyed=set(), done={'N01'}
[DSP] Composant N02 bloqué définitivement après échec injecté (DSP strict)
[INFO] Étape 2 : N05 (normal)
[INFO] État overlay : locked={'N11', 'N06', 'N02', 'N17'}, destroyed=set(), done={'N01', 'N05'}
[DEBUG] detect_failure (injection) pour N10 => FailureEvent(comp_id='N10', op_type='unscrew', tool_id='T_hex', symptom=<Sym
[INFO] Injection d'échec sur le composant : N10 (étape 3)
[INFO] FailureEvent détecté : {"symptom": "TIMEOUT", "signals": {"timeout": 1}, "context": {"op_type": "unscrew", "tool_id"
[DEBUG] FailureEvent objet : FailureEvent(comp_id='N10', op_type='unscrew', tool_id='T_hex', symptom=<Symptom.TIMEOUT: 5>,
[Decision] action=replan
[INFO] Résultat action : success=True, updates={'new_plan': ['N10', 'N03', 'N07', 'N16', 'N04', 'N12', 'N20', 'N08', 'N09',
[INFO] État overlay : locked={'N11', 'N06', 'N02', 'N17'}, destroyed=set(), done={'N01', 'N05'}
[DSP] Composant N10 bloqué définitivement après échec injecté (DSP strict)
[ACTIONS] replan_grasp: new_plan=['N10', 'N03', 'N07', 'N16', 'N04', 'N12', 'N20', 'N08', 'N09', 'N14', 'N13', 'N15', 'N19'
[INFO] Nouveau plan après replan : ['N10', 'N03', 'N07', 'N16', 'N04', 'N12', 'N20', 'N08', 'N09', 'N14', 'N13', 'N15', 'N1
[INFO] Étape 4 : N03 (normal)
[INFO] État overlay : locked={'N11', 'N16', 'N20', 'N02', 'N10', 'N17', 'N06'}, destroyed=set(), done={'N03', 'N01', 'N05'}
```

```
[INFO] État overlay : locked={'N11', 'N16', 'N20', 'N02', 'N10', 'N17', 'N06'}, destroyed=set(), done={'N03', 'N21', 'N13', 'N09', 'N19', 'N07',
[INFO] Étape 16 : N22 (normal)
[INFO] État overlay : locked={'N11', 'N16', 'N20', 'N02', 'N10', 'N17', 'N06'}, destroyed=set(), done={'N03', 'N21', 'N13', 'N09', 'N19', 'N07',
[INFO] Plan effectif vide, arrêt de la boucle principale.
[INFO] État final overlay : locked={'N11', 'N16', 'N20', 'N02', 'N10', 'N17', 'N06'}, destroyed=set(), done={'N03', 'N21', 'N13', 'N09', 'N19',
[TRACE] Plan initial : ['N01', 'N02', 'N05', 'N06', 'N10', 'N03', 'N11', 'N07', 'N04', 'N09', 'N16', 'N08', 'N12', 'N13', 'N17', 'N18', 'N14', '
[TRACE] Plan effectif (après adaptation) : []
[TRACE] Séquence réellement suivie : ['N01', 'N05', 'N03', 'N07', 'N04', 'N12', 'N08', 'N09', 'N14', 'N13', 'N15', 'N19', 'N18', 'N21', 'N22']
[VALIDATION DSP] Séquence réellement suivie VALIDE (DAG, règles d'ordre, cible(s), pas de doublon)
[BILAN FINAL] Séquence terminée.
[BILAN FINAL] Cibles atteintes : ['N22']
[BILAN FINAL] Cibles inatteignables : []
[BILAN FINAL] Robustesse du plan : 1/1 cibles atteintes.
```

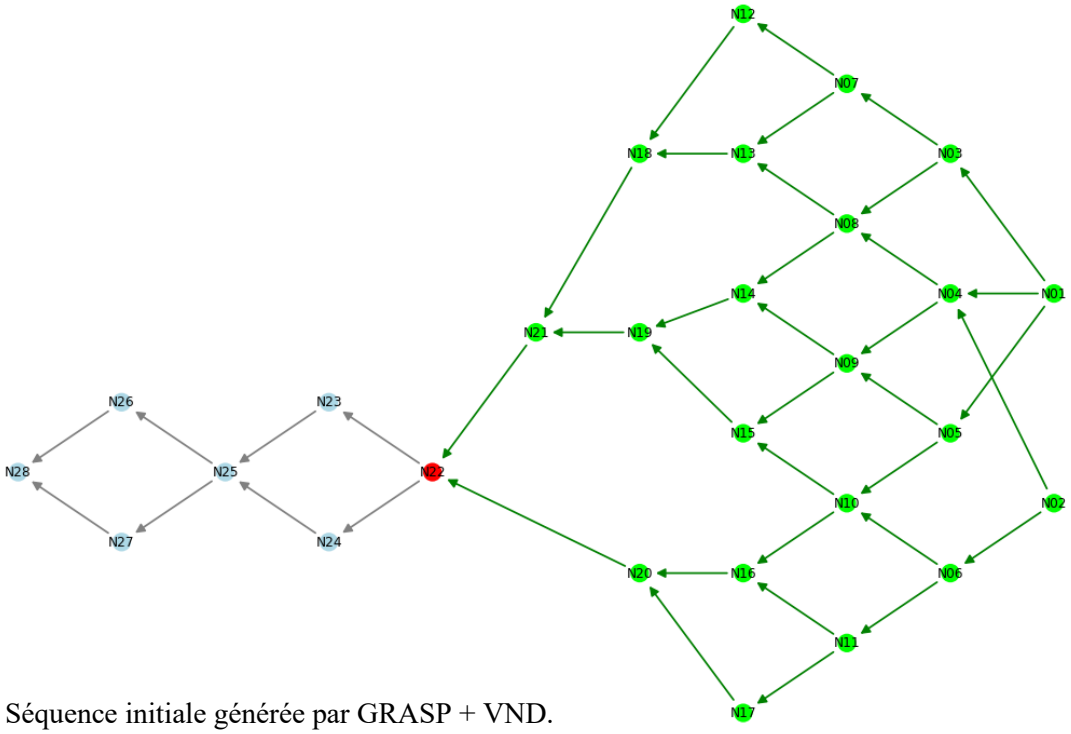
Résumé JSON stress test adaptif :

results > stress_logs > dagtest > stress_log_failures_3_on_dagtest.json > ...


```
{
  "plan_initial": [
    "N01",
    "N02",
    "N05",
    "N06",
    "N10",
    "N03",
    "N11",
    "N07",
    "N04",
    "N09",
    "N16",
    "N08",
    "N12",
    "N13",
    "N17",
    "N18",
    "N14",
    "N15",
    "N20",
    "N19",
    "N21",
    "N22"
  ],
  "plan_effectif": [],
  "failures": "data/adaptive/generated_failures/dagtest/failures_3_on_dagtest.json",
  "overlay_final": {
    "locked": [
      "N11",
      "N16",
      "N20",
      "N02",
      "N10",
      "N17",
      "N06"
    ],
    "destroyed": [],
    "done": [
      "N03",
      "N21",
      "N13",
      "N09",
      "N19",
      "N07",
      "N14",
      "N18",
      "N04",
      "N15",
      "N08",
      "N22",
      "N01",
      "N05",
      "N12"
    ]
  },
  "targets": [
    "N22"
  ],
  "targets_status": {
    "N22": true
  },
  "steps": [
    {
      "comp": "N01",
      "step": 1,
      "action": "normal",
      "done": true,
      "notes": null
    },
    {
      "comp": "N05",
      "step": 2,
      "action": "normal",
      "done": true,
      "notes": null
    },
    {
      "comp": "N10",
      "step": 3,
      "action": "replan",
      "done": false,
      "notes": "Replanification réussie."
    },
    {
      "comp": "N03",
      "step": 4,
      "action": "normal",
      "done": true,
      "notes": null
    },
    {
      "comp": "N07",
      "step": 5,
      "action": "normal",
      "done": true,
      "notes": null
    },
```

Visuels du graphe complet de l'instance :

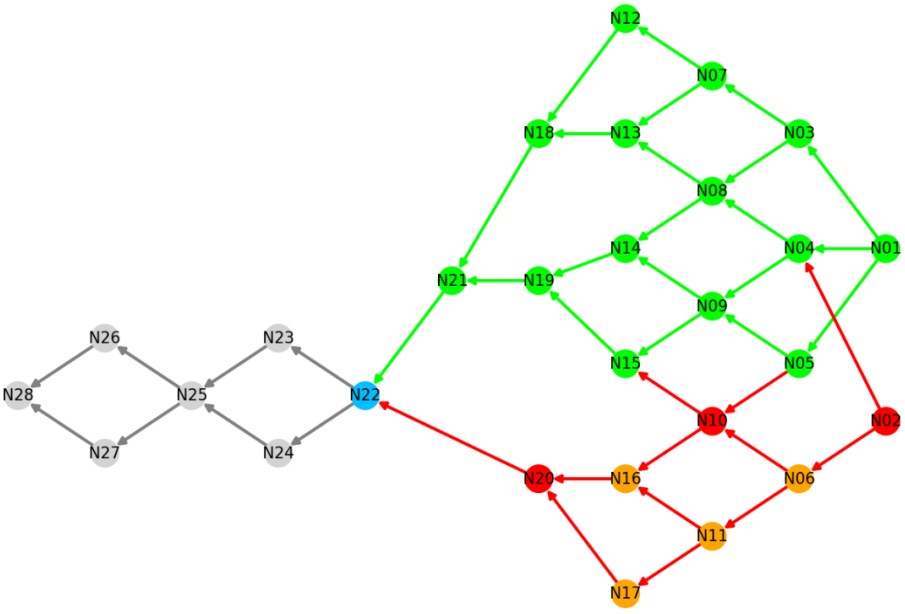
Séquence initiale (avant échec) :



Séquence initiale générée par GRASP + VND.

Les nœuds verts représentent les composants actifs, tandis que les arêtes indiquent les contraintes de précédence entre opérations.

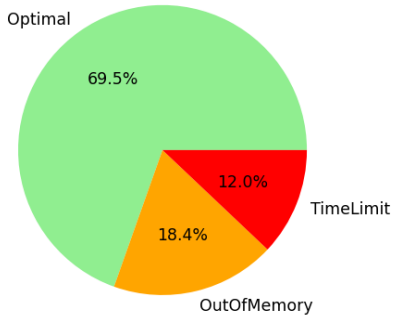
Séquence replanifiée (après échec) :



Les nœuds rouges indiquent les composants en échec, orange les zones impactées, et bleu la cible atteinte (**N22**).

Résultats globaux du pipeline (sur plusieurs instances) :

Répartition des statuts de résolution globaux :



69.5 % des instances atteignent la solution optimale,

18.4 % s'arrêtent sur OutOfMemory,

12.0 % sur TimeLimit.

Extrait des résultats détaillés des instances testées (extrait du tableau Excel) :

instance	status	gap_milp	time_milp	profit_milp	gap_grasp	time_grasp	profit_grasp
salbp_instance_n=50_53p3	Optimal	0	0,362757206	332	0	0,013266325	332
salbp_instance_n=50_53p4	Optimal	0	0,346890926	174	0	0,008653402	174
salbp_instance_n=50_53p5	Optimal	0	0,338111401	504	0	0,017603636	504
scholl_arc111_n=111	Optimal	0	69,85451627	2131	0	0,154135227	2131
scholl_arc83_n=83	Optimal	0	3,839942217	1594	0	0,096289873	1594
scholl_barthol2_n=148	Optimal	0	505,7292774	21			
scholl_bowman8_n=8	Optimal	0	0,017223597	92	0	0,004431963	92
scholl_buxey_n=29	Optimal	0	0,123231173	519	0	0,017357111	519
scholl_gunther_n=35	Optimal	0	0,176827669	581	0	0,020074844	581
scholl_hahn_n=53	Optimal	0	1,164047718	1100	0	0,053789616	1100
scholl_heskia_n=28	Optimal	0	0,069168806	525	0	0,017258883	525
scholl_jackson_n=11	Optimal	0	0,015756369	177	0	0,007301807	177
scholl_jaeschke_n=9	Optimal	0	0,016411304	145	0	0,00629425	145
scholl_kilbrid_n=45	Optimal	0	1,078080654	796	0	0,028481483	796
scholl_lutz1_n=32	Optimal	0	0,126522779	631	0	0,020700932	631
scholl_lutz2_n=89	Optimal	0	6,458083391	1509	0	0,102458	1509
scholl_lutz3_n=89	Optimal	0	6,804907084	1689	0	0,104358196	1689
scholl_mansoor_n=11	Optimal	0	0,014756441	215	0	0,007042646	215
scholl_mertens_n=7	Optimal	0	0,063465595	127	0	0,005175114	127
scholl_mitchell_n=21	Optimal	0	0,035233498	331	0	0,012425184	331
scholl_mukherje_n=94	Optimal	0	56,17790675	1856	0	0,110516787	1856
scholl_roszieg_n=25	Optimal	0	0,074919701	369	0	0,014294863	369
scholl_sawyer30_n=30	Optimal	0	0,250006437	65	0	0,00618434	65
scholl_tonge70_n=70	Optimal	0	14,87166762	830	0	0,038268566	830
scholl_warnecke_n=58	Optimal	0	7,724807978	45	0	0,007522345	45
salbp_instance_n=1000_172_cut361	TimeLimit	29,4	1849,975427	216	0	0,037910223	216
salbp_instance_n=1000_175_cut259	TimeLimit	0,52	1819,878333	1022	0	0,062856436	1022
salbp_instance_n=1000_19_cut280	TimeLimit	1,09	1824,106487	434	0	0,039461851	434
salbp_instance_n=1000_1_cut244	TimeLimit	18,24	1816,680221	115	0	0,02294445	115
salbp_instance_n=1000_1_cut245	TimeLimit	1,02	1817,505366	290	0	0,029583693	290
salbp_instance_n=1000_1_cut246	TimeLimit	18,14	1817,381056	316	0	0,032954931	316
salbp_instance_n=1000_1_cut254	TimeLimit	0,45	1818,569228	567	0	0,042618036	567
salbp_instance_n=1000_1_cut271	TimeLimit	37,36	1822,714904	380	0	0,040049076	380
salbp_instance_n=1000_1_cut291	TimeLimit	37,63	1828,628531	328	0	0,035983324	328
salbp_instance_n=1000_1_cut625	OutOfMemory		0			0,082593679	496
salbp_instance_n=1000_1_cut637	OutOfMemory		0			0,076367617	871
salbp_instance_n=1000_1_cut639	OutOfMemory		0			0,066166162	373
salbp_instance_n=1000_1_cut644	OutOfMemory		0			0,109980106	1394
salbp_instance_n=1000_1_cut654	OutOfMemory		0			0,13867712	1914
salbp_instance_n=1000_1_cut664	OutOfMemory		0			0,173889399	2255
salbp_instance_n=1000_1_cut671	OutOfMemory		0			0,378473759	3598
salbp_instance_n=1000_1_cut675	OutOfMemory		0			0,182599068	2347
salbp_instance_n=1000_1_cut676	OutOfMemory		0			0,268614769	2902
salbp_instance_n=1000_1_cut684	OutOfMemory		0			0,119740725	1454
salbp_instance_n=1000_1_cut686	OutOfMemory		0			0,15248704	1862