

Configurateur de PC avec CSP

Nassim Lattab - Université Paris Cité

Février 2025

Comment garantir la compatibilité des composants d'un PC tout en facilitant la sélection ?

- **Complexité** : Beaucoup de composants avec des spécifications variées.
- **Contraintes techniques** : Compatibilité entre CPU, RAM, GPU, etc.
- **Budget** : Optimisation des coûts.

Solution : Un configurateur intelligent qui...

- Vérifie automatiquement la compatibilité des composants.
- Permet une sélection interactive optimisée.
- Intègre des contraintes budgétaires plus globales.

Deux stratégies principales :

- **Approche avec Solveur CSP**

- Génère toutes les solutions possibles avant de filtrer.
- Assure une solution valide mais peut être lent sur de grands ensembles.
- Facilite l'intégration des contraintes globales

- **Approche sans Solveur (MAC - Propagation des Contraintes)**

- Sélection d'une solution en temps réel.
- Réduit progressivement les domaines sans générer toutes les combinaisons.
- Moins efficace pour gérer les contraintes budgétaires globales.

Quelles contraintes doivent être respectées ?

● **Compatibilité matérielle (contraintes locales)**

- CPU \leftrightarrow Carte mère (même socket)
- Carte mère \leftrightarrow RAM (même type de mémoire)
- Carte mère \leftrightarrow Boîtier (format supporté)
- GPU \leftrightarrow Alimentation (puissance suffisante)
- Alimentation \leftrightarrow Boîtier (format supporté)

● **Contraintes globales**

- Respect du budget imposé (pour l'approche avec solver)*

**La contrainte de budget est difficile à appliquer avec MAC, car les choix sont faits progressivement sans visibilité sur le coût total final. Il n'est pas possible de garantir que les sélections intermédiaires permettront d'atteindre une solution respectant une contrainte budgétaire globale.*

Qu'est-ce que python-constraint ?

- Bibliothèque pour résoudre des problèmes de satisfaction de contraintes (CSP).
- On définit des variables, des domaines et des contraintes.
- Recherche automatique des solutions valides.

Étapes :

- ① Déclaration d'un problème avec `Problem()`.
- ② Ajout des variables et de leurs domaines.
- ③ Définition des contraintes.
- ④ Recherche de solutions avec `getSolutions()` ou `getSolution()`.

Comparaison entre getSolutions() et getSolution()

Quelle méthode utiliser ?

Critère	getSolutions()	getSolution()
Retour	Toutes les solutions possibles	Une seule solution valide
Performance	Lent si trop de solutions	Rapide et efficace
Mémoire utilisée	Élevée (stocke toutes les solutions)	Faible (retourne une seule solution)
Quand l'utiliser ?	Pour comparer plusieurs options	Si une seule solution suffit
Optimisation	Nécessite un tri des résultats	Pas forcément optimale

Approche avec Solveur vs Approche sans Solveur (MAC)

Critère	Avec Solveur CSP	Sans Solveur (MAC)
Principe	Génère toutes les solutions	Sélection progressive
Scalabilité	Lent avec grands ensembles	Efficace pour gros jeux de données
Interaction	Affine la solution selon l'ensemble des solutions	Affine la solution en temps réel
Complexité	Exponentiel dans le pire des cas	Réduction progressive de l'espace de recherche
Explicabilité	Permet de calculer toutes les solutions possibles	Affiche les options compatibles étape par étape
Contraintes budget	Facile à implémenter	Plus complexe (coût connu tardivement)

Pourquoi ne pas combiner les forces des deux méthodes ?

- **MAC** pour filtrer en amont et réduire l'espace de recherche.
- **Solveur** pour trouver la meilleure solution dans un sous-ensemble réduit (utile pour des modèles multi-critères).
- **Filtrage dynamique** : Résolution locale dès qu'une contrainte critique est fixée.

Analyse des stratégies de résolution par contraintes :

- Les approches CSP permettent différentes stratégies : exploration exhaustive (Solveur CSP) ou propagation des contraintes (MAC).
- MAC permet de maintenir la consistance locale en fixant les composants un à un, évitant le backtracking.
- L'approche avec Solveur CSP est particulièrement utile lorsque plusieurs solutions doivent être générées.
- MAC est plus efficace lorsqu'une seule solution valide est nécessaire, car il restreint dynamiquement l'espace de recherche sans explorer toutes les possibilités.
- Dans un cadre d'optimisation multi-critères, le solveur CSP permet d'obtenir directement une ou plusieurs solutions optimisées.
- Le solveur CSP facilite l'intégration des contraintes globales en filtrant directement les solutions non valides dès la génération.

Améliorations possibles :

- Ajout de nouveaux composants pour complexifier les contraintes (ex. ventilateurs, watercooling, stockage disque, ssd...)
- Intégration d'un moteur multi-critères pour optimiser selon plusieurs contraintes globales (coût, volume sonore, consommation énergétique...).
- Recherche d'heuristiques pour améliorer la sélection des composants.
- Étude d'approches hybrides associant la propagation des contraintes et des heuristiques d'optimisation.