Métaheuristiques et contraintes

Jin-Kao Hao

LERIA, Université d'Angers

Plan

1) Rappels
   a) Problème de satisfaction de contraintes (CSP) et exemples (k-coloration et carré latin).
   b) Problème d'optimisation sous contraintes (COP) et exemple (coloration pondérée).
   c) Métaheuristiques pour l'optimisation combinatoire (recherche locale et méthodes à population).
2) Métaheuristiques pour la résolution du CSP
   a) Recherche locale : fonction d'évaluation à base de pénalités et voisinage.
   b) Méthode mémétique : croisement et population.
   c) Exemples de k-coloration et achèvement du carré latin.
3) Métaheuristiques pour la résolution du COP
   a) Stratégies pour le traitement de contraintes.
   b) Fonction d'évaluation étendue et voisinage.
   c) Exemple de coloration pondérée.

Optimisation combinatoire et problèmes de contraintes

Problème d'optimisation combinatoire

Etant donné un couple (Ω, f) avec

- Ω : un ensemble fini de *solutions* ou *configurations* (*espace de recherche*)
- $f : \Omega \rightarrow R$ une *fonction de coût* ou *objectif*,

trouver une configuration *s** de Ω qui minimise (maximise) f.

CSP (Constraint satisfaction problem)

Etant donné un un réseau de contraintes (X, D, C) où

- $X = \{x_1, \dots, x_n\}$ ensemble de variables
- $D = \{D_1, \dots, D_n\}$ collection de domaines de valeurs associés aux variables
- $C = \{C_1, \dots, C_m\}$ ensemble de contraintes portant sur des sous-ensembles de variables

Le CSP consiste à déterminer une instanciation des variables par des valeurs prises dans les domaines pour satisfaire toutes les contraintes.

COP (Constrained optimisation problem)

Etant donné un réseaux de contraintes (X, D, C) et une fonction de coût ou objectif f à optimiser (minimiser ou maximiser), le COP consiste à déterminer une instanciation des variables pour optimiser la fonction f tout en satisfaisant toutes les contraintes.

Optimisation combinatoire et problèmes de contraintes

- Le CSP n'est pas un problème d'optimisation en soi, mais peut être vu comme un problème d'optimisation combinatoire (Ω, f) (Max-CSP) où
  - Ω : ensemble des instanciations des variables cohérentes et incohérentes.
  - f : fonction de minimisation qui compte le nombre de contraintes non-satisfaites.

  Ainsi, le CSP peut être traité par des méthodes d'optimisation comme les métaheuristiques
- CSP et COP sont des modèles génériques pour formuler de nombreux problèmes et applications.
- La plupart des problèmes combinatoires les plus intéressants (y compris CSP et COP) sont NP-difficiles (il n'existe pas d'algorithmes polynomiaux sauf si P=NP).

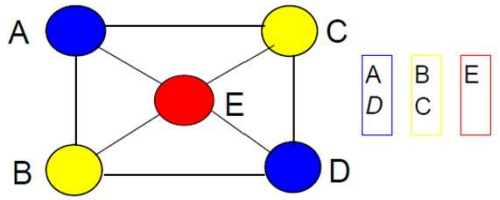
  Ainsi, des méthodes approchées tq les métaheuristiques sont nécessaires.

Exemple CSP – *k*-coloration

k-coloration (k-COL) : Soit $G = (V, E)$, et k couleurs, colorier les sommets de G avec ces k couleurs tel que deux sommets adjacents ne reçoivent pas la même couleur (contrainte de coloration). Si une telle coloration existe, G est dit *k-coloriable* (cf. exemple).

RMQ : Une k-coloration peut être

- définie par une fonction $c : V \to \{1 \dots k\}$ tq $\forall \{x, y\} \in E, c(x) \neq c(y)$.
- ou considérée comme une *partition* des sommets de V dans k classes de couleurs (stables) où chaque classe de couleurs regroupe les sommets ayant reçu la même couleur.



Le **problème de coloration (COL)** est de déterminer le nombre chromatique de $G, \chi(G)$, i. e. , le nombre minimum de couleurs k tq G est $k-$coloriable pour un graphe arbitraire.

RMQ:

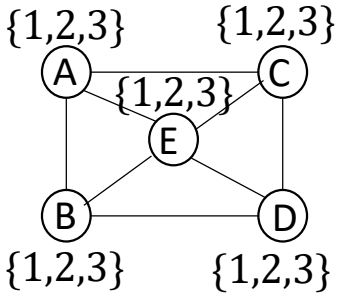
- k-COL est NP-complet (NP-difficile pour COL).
- La résolution successive du problème de k-coloration (avec $k-1, k-2 \dots$) permet de déterminer une borne supérieure du nombre chromatique.
- k-COL et COL ont de nombreuses applications pratiques : affectation de fréquence, emploi du temps…

Exemple CSP – *k*-coloration

k-coloration : Soit $G = (V, E)$, et k couleurs, colorier les sommets de G avec ces k couleurs tq deux sommets adjacents ne reçoivent pas la même couleur.

Problème de k-coloration = un CSP particulier

- X: une variable (entière) pour un sommet
- D: $D_1 = \cdots = D_n = \{1,2, \dots, k\}$ (les k couleurs)
- C: chaque arête $\{x, y\} \in E \Leftrightarrow$ une contrainte binaire : $x \neq y$.



Déterminer une k-coloration ⇔ résoudre le CSP (i.e., trouver une instanciation des variables par des valeurs $\{1,2, \dots, k\}$ pour satisfaire toutes les contraintes).

Exemple CSP – carré latin

Un **carré latin** *L* d'ordre *n* (ou Quasigroup) est un tableau *n* x *n* rempli de *n* symboles distincts (e.g., {1…,*n*}) tq chaque symbole apparaît exactement une fois sur chaque ligne et chaque colonne (contrainte de carré latin)

1	3	2
3	2	1
2	1	3

Un carré Latin = une solution du CSP (*X,D,C*) suivant

- X: une variable (entière) correspond à une cellule de *L* ($|X| = n^2 = N$)
- D: $D_i = \{1, \dots, n\}$ $(i = 1, \dots, N)$
- C: *alldifferent*$(x_1 \dots x_n)$ pour chaque ligne et chaque colonne; i. e. , $\forall x, y \in X, x \neq y$ si x et y représentent deux cellules d'une même ligne ou colonne.

Soit *L* un carré latin d'ordre *n* partiellement rempli, le problème « **LSC - Latin square completion** ou **Achèvement du carré latin** » consiste à compléter les cellules vides de *L* pour obtenir un carré latin.

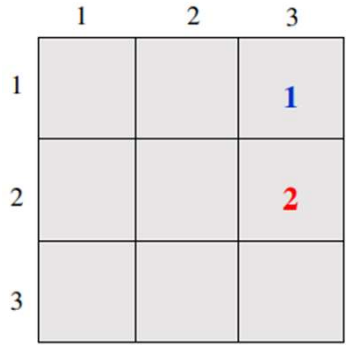
RMQ :

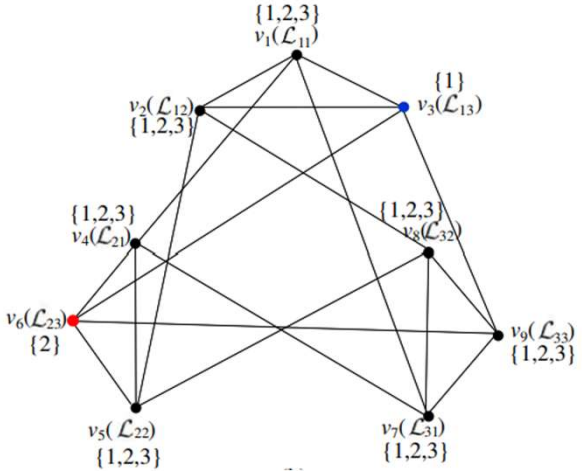
- LSC est un CSP et difficile à résoudre (NP-complet).
- LSC est un cas particulier du problème d'optimisation « *PLSE- Partial Latin square extension* ou *Extension du carré latin partiel* » qui consiste à compléter le maximum des cellules vides de *L*.

Achèvement du carré latin *vs* k-coloration

Conversion d'un carré latin en graphe (graphe carré latin) :

Soit *L* un carré latin d'ordre n, construire un graphe carré latin $G = (V, E)$ où chaque sommet représente une cellule de *L* ($|V| = n^2$) et $\{u, v\} \in E$ ssi u et v représentent deux cellules d'une même ligne ou colonne ($|E| = n^2(n-1)$).





CSP

- $X = \{v_1, \dots, v_9\}$
- $D_3 = \{1\}, D_6 = \{2\}, D_i = \{1,2,3\}$ $(i \neq 3,6)$
- C : *alldifferent*(…) pour chaque ligne - colonne

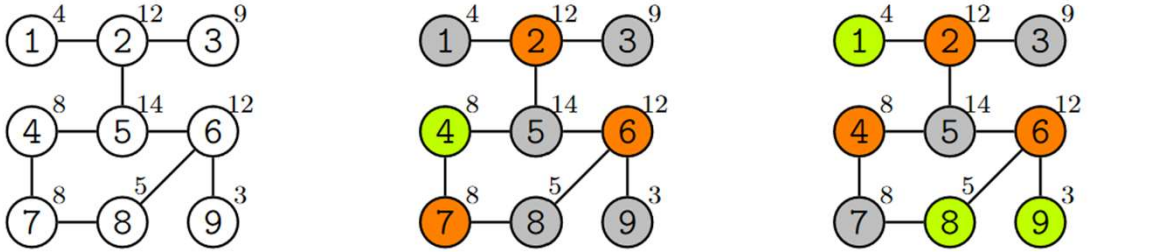
Alors une solution à l'instance *L* du problème LSC « **Achèvement du carré latin** » correspond à une solution du CSP ou une *k*-coloration du graphe carré latin où certains sommets ont déjà reçu une couleur.

Exemple COP – coloration pondérée

Coloration pondérée (WVCP - weight vertex coloring) : Soit graphe $G = (V, E)$ ($|V| = n$), et $W = \{w_1, \dots, w_n\}$ l'ensemble des poids associés aux sommets de *V*.

Déterminer une coloration légale, i.e., *k* classes de couleurs $\{V_1, \dots, V_k\}$, pour minimiser le coût total des *k* classes de couleurs, le coût d'une classe étant le plus grand poids de cette classe.

$$(WVCP) \quad \text{minimize} \quad f(s) = \sum_{i=1}^{k} max_{j \in V_i} w_j$$



(a) A graph $G = (V, E)$ (b) A feasible coloring (c) An optimal solution

Colorations (b) et (c) ont 3 classes de couleurs: gris, orange et vert avec valeur de coût *f* :
(b): 14 (gris) + 12 (orange) + 8 (vert) = 34
(c): 14 (gris) + 12 (orange) + 5 (vert) = 31

RMQ:
1) WVCP est un COP avec l'objectif de minimisation *f* sous contraintes de coloration.
2) WVCP généralise COL : quand les poids sont à 1, WVCP devient COL.

Métaheuristiques

- Une **métaheuristique** est une **méthode d'optimisation approchée**, **générale** et applicable à différents problèmes, contrairement à une **heuristique** qui est conçue pour un *problème particulier*.
- Les **métaheuristiques**
  - visent à trouver des solutions de qualité (non nécessairement optimales) en temps raisonnable (polynomial) pour des problèmes d'optimisation combinatoire complexes ;
  - permettent aussi de résoudre d'autres problèmes combinatoires (e.g., CSP).
- Champs d'applications privilégiés : problèmes combinatoires dont la solution optimale est difficile à obtenir ou l'optimalité n'est pas primordiale.

RMQ:

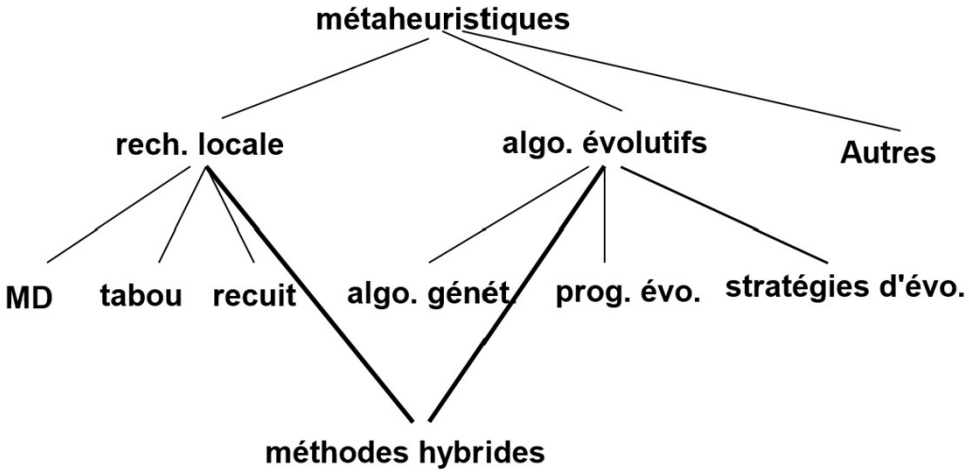
- L'application d'une métaheuristique à un problème particulier nécessite des adaptations adéquates au problème traité (tout comme pour toute méthode d'optimisation générale tq B&B, PD…).

Métaheuristiques - panorama

La recherche locale échantillonne itérativement des solutions de l'espace de recherche Ω à l'aide de trois éléments de base : *voisinage*, *fonction d'évaluation* et *règle de transition.*

Une métaheuristique à population utilise une ensemble de solutions (*population*) pour échantillonner l'espace de recherche Ω à l'aide d'une *fonction d'évaluation* et des *opérateurs d'évolution (sélection, croisement, mutation).*

Métaheuristique mémétique (méthode à population + recherche locale) utilise une recherche locale pour améliorer chaque nouvelle solution issue du croisement.



RMQ:

- La recherche locale et la méthode mémétique sont deux approches très populaires pour l'optimisation difficile.

Métaheuristiques – éléments de base

Recherche locale

- Le **voisinage** N est une fonction $\mathrm{N} : \Omega \rightarrow 2^{\Omega}$ qui associe à chaque solution s un sous-ensemble N(s) de Ω (l'ensemble des voisins de s). Il est typiquement défini par un opérateur de **mouvement** qui modifie (légèrement) la solution courante.
- La **fonction d'évaluation** F mesure la qualité d'une solution s. Elle typiquement correspond à la fonction de coût *f*, mais peut être étendue.

Méthode à population

- La **population** maintient un ensemble de solutions de **qualité et diversifiées** qui participent à la découverte de nouvelles solutions.
- La **fonction d'évaluation (fitness)** F mesure la qualité d'une solution s.
- Le **croisement** est un mécanisme de recombinaison de deux ou plusieurs solutions existantes (**parents**) pour générer de nouvelles solutions (**enfants**).
- La **distance** est utilisée pour gérer la diversité de la population.

Recherche locale pour la résolution du CSP

Espace de recherche Ω

- Une configuration s de Ω est une instanciation complète des variables (pas nécessairement cohérente).
- Ex. k-coloration : Ω est composé de toutes les colorations des sommets avec k couleurs, y compris celles avec conflits (i.e., toutes les partitions des sommets en k classes de couleurs).

Fonction d'évaluation (à minimiser)

- F(s) mesure la violation des contraintes dans une solution s.
  - Dans le cas général, F(s) est définie par une fonction de pénalité (voir page suivante).
  - Dans les cas simples, F(s) compte le *nombre de contraintes violées* dans s.
  - Ex. k-coloration (cas simple) : F(s) compte le nombre de conflits dans une coloration s (un conflit = une contrainte de coloration violée = une arête dont les extrémités ont reçu la même couleur).

Voisinage (le plus utilisé)

- Il est typiquement défini par le mouvement « **one-move <x, v>** » qui change la valeur courante s(x) de la variable x par une nouvelle valeur $v \in D_x$.
  - Ex. k-coloration (cas simple) : on change la couleur d'un sommet.
  - Le choix de la variable x à changer peut être soumis à des conditions spécifiques (voir plus loin).
- D'autres mouvements sont possibles selon la représentation des solutions (voir plus loin).

Recherche locale pour le CSP – fonction d'évaluation

Principe d'une fonction de pénalité

- **Fonction de pénalité f_c associée à une contrainte C**
  Si la configuration s satisfait C, alors $f_c(s) = 0$; sinon $f_c(s) > 0$.

- **Fonction d'évaluation globale F (à minimiser)**
  Elle est définie par la somme pondérée des fonctions de pénalité des contraintes

$$F(s) = \sum w_c * f_c(s)$$

où w_c est un coefficient (poids) associé à contrainte C, indiquant l'importance relative accordée à contrainte C.

Si $F(s) = 0$ (toutes les contraintes sont satisfaites), alors s est une solution du CSP.

Recherche locale pour le CSP – fonction d'évaluation

Comment définir une fonction de pénalité

- **Fonction de pénalité f_c associée à une *contrainte simple***

  Si la configuration s satisfait C, $f_c(s) = 0$; sinon $f_c(s) = 1$

  Exemples :

  - *different*(*x, y*) ($x \neq y$): $f_C(s) = 0/1$
  - *distance*(*x, y, D*) ($|x-y|>D$): $f_C(s) = 0/1$

- **Fonction d'évaluation f_c associée à une *contrainte complexe***

  $f_c(s)$ mesure le **dégrée de violation** de la contrainte C dans s

  Exemples (contraintes globales) :

  - *alldifferent*($[y_1..y_k]$) : $f_C(s) = 0$ ou le nombre de pairs $(y_i = y_j)$ $\forall\ i \neq j$ qui violent la contrainte.
  - *atmost*(*L*, $[y_1..y_k]$, a) (le nb de "$y_i = a$" $\leq L$) : $f_C(s) = 0$ ou le dépassement (le nombre de "$y_i = a$" – L).
  - *atleast*(*L*, $[y_1..y_k]$, a) (le nb de "$y_i = a$" $> L$) : $f_C(s) = 0$ ou le manquement (L – le nombre de "$y_i = a$").
  - *capa*($[y_1..y_k]$, a, $[w_1..w_k]$, *W*) (le poids total des variables y_i prenant la valeur a $\leq$ W) : $f_C(s) = 0$ ou le dépassement de la capacité *W*.

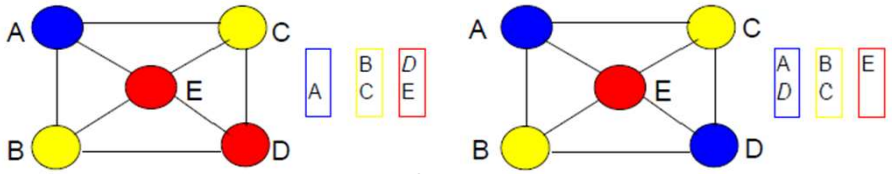
Recherche locale pour le CSP – voisinage

Voisinage « one-move »

- Le **one-move <x, v>** consiste à donner à x la valeur v ∈ D_x – {s(x)} à la place de sa valeur courante s(x).
- Le **voisinage « one-move »** contient toutes les configurations générées par one-move <x,v> pour tout x ∈ X et v ∈ D_x - {s(x)} appliqué à la configuration s.

Variable critique et voisinage réduit

- Une **variable** est **critique** si elle est impliquée dans une contrainte **violée** par la configuration s.
- Le **voisinage réduit** comprend les solutions générées par « one-move » appliqué aux variables critiques.
- Ex. k-coloration : une variable critique = un sommet en conflit ; on s'intéresse uniquement aux sommets en conflit pour le changement de couleur.



RMQ:

- Le voisinage réduit est de plus petite taille et favorise la minimisation des conflits.
- Pour la k-coloration, le voisinage réduit a une taille de nc×(k-1) (nc = le nombre de sommets en conflit), qui est largement plus petite que le voisinage complet dont la taille est de n×(k-1) (n = le nombre des sommets).

Recherche locale pour le CSP - voisinage

Autres voisinages pour le CSP selon la représentation des solutions

- Représentation de type « **binaire** » (sac-à-doc, clique, sélection de sites…)
  Mouvements : flip (add / drop), swap, k-flip…
- Représentation de type « **permutation** » (TSP, VRP, étiquetage de graphes, ordonnancement,…)
  Mouvements : swap, rotation, insertion...
- Représentation type « **entier** » (coloration…)
  Mouvements : one-move, swap…

RMQ:

- Puisque le but est de satisfaire les contraintes, il est souvent pertinent d'identifier des critères pour favoriser la résolution des conflits et restreindre le voisinage (e.g., les mouvements portant sur des variables en conflit).

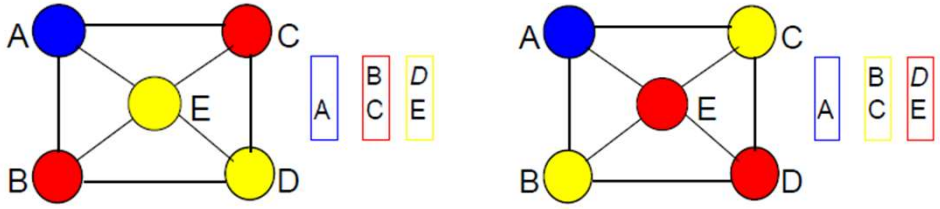
Approche mémétique pour le CSP - croisement

Croisement (ou recombinaison)

- Principe : construire de nouvelles solutions à partir de deux ou plusieurs solutions existantes.
- Stratégie pour un croisement pertinent :
  a) Analyser le problème et identifier les caractéristiques « favorables » (building blocks) dans des solutions élites ;
  b) Conserver ces caractéristiques via le mécanisme de recombinaison.

Exemple de k-coloration

- Ce qui caractérise une coloration, c'est les **groupements de sommets** qui peuvent recevoir la même couleur (non pas la couleur en soi) (cf. l'exemple suivant).
  Une caractéristique « favorable » concerne les grandes classes de couleur.



Permuter les couleurs des classes donne la même coloration

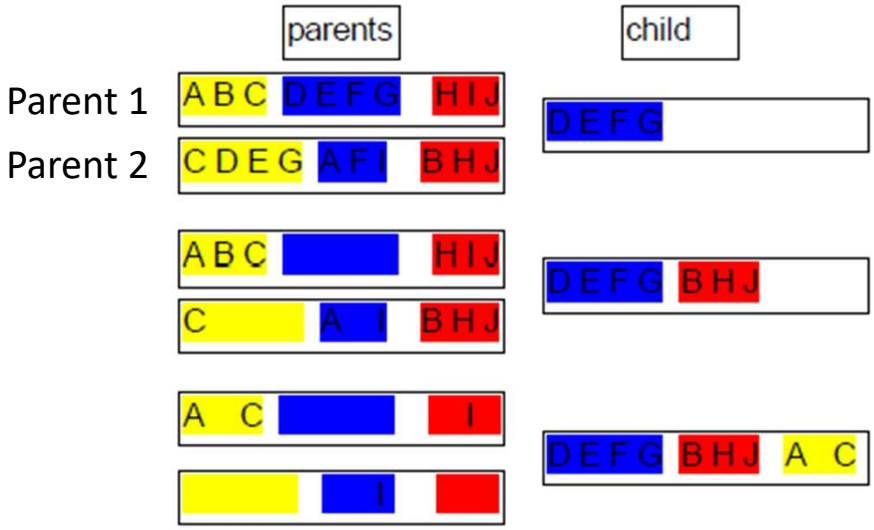
- Le croisement doit conserver dans l'enfant les grandes classes de couleur des parents.

Approche mémétique pour le CSP - croisement

Exemple de k-coloration.

Le « GPX - Greedy Partition Crossover » met en place cette idée.

- Transmettre à l'enfant les plus grandes classes de couleur des parents.
- Pour équilibre la transmission, alterner la transmission entre les deux parents.



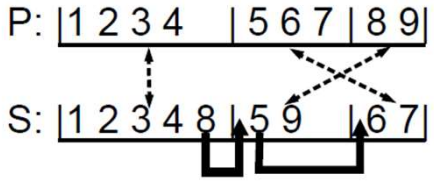
Approche mémétique pour le CSP - population

Population

- Principe : maintenir un ensemble de solutions à la fois de **qualité et diversifiées**.
- Stratégie pour gérer la population :
  - Utiliser une **métrique de distance appropriée** pour garantir une séparation minimum entre les solutions de la population pour éviter une convergence prématurée de la recherche.
  - Lors de la mise à jour de la population avec une nouvelle solution, prendre en compte à la fois la qualité et la distance.

Exemple de k-coloration

- Une métrique pertinente pour mesurer la dissimilarité (distance) entre deux colorations P et S est la distance de partition ou de transfert, qui est égale au nombre minimum de one-moves nécessaires pour transformer S en P (ou P en S).



La distance entre P et S = 2
(déplacer sommets 8 et 5)

Approche mémétique pour le CSP – application à k-coloration

L'espace de recherche Ω est composé de toutes les partitions des sommets en k classes de couleurs, y compris celles avec conflits (i.e., colorations légales et illégales).

La fonction d'évaluation F(s) compte le nombre de conflits dans s (i.e., nombre de contraintes de coloration violées par s).

Le croisement GPX avec deux parents (ou plusieurs parents).

Le voisinage réduit avec le mouvement « one-move » appliqué aux sommets **en conflit**.

Recherche local avec la métaheuristique **Tabu Search**

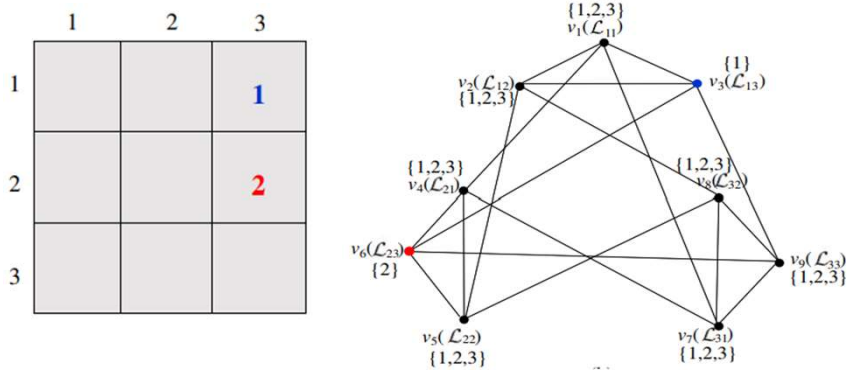
- Chaque itération de l'algorithme utilise le meilleur voisin pour remplacer la solution courante (i.e., choisir le mouvement <x, v> le plus favorable qui donne le meilleur gain en terme de conflits).
- Chaque mouvement <x, v> effectué est stocké dans une liste taboue et il est interdit de changer la couleur du sommet x pendant β itérations (β est une fonction du nombre de conflits).

RMQ :

- Cette approche est parmi les approches les performantes pour résoudre les benchmarks DIMACS.

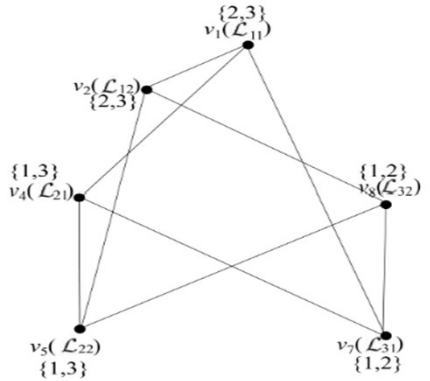
Approche mémétique pour le CSP – application au LSC

La même approche appliquée au problème « LSC - **achèvement du carré latin** » : compléter un carré latin *L* d'ordre *n* partiellement rempli avec *n* symboles distincts (e.g., {1…,*n*}) tq chaque symbole apparaît exactement une fois sur chaque ligne et chaque colonne.



CSP

- $X = \{v_1, \dots, v_9\}$
- $D_3 = \{1\}, D_6 = \{2\}, D_i = \{1,2,3\}\ (i \neq 3,6)$
- C : *alldifferent*(…) pour chaque ligne - colonne



Approche de résolution

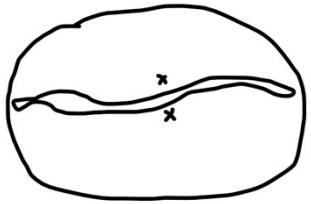
- Appliquer en pre-processing la **propagation de contraintes** pour réduire les domaines à partir des cellules pré-remplies, donnant un graphe (CSP) réduit.
- K-colorier le graphe réduit avec **l'algorithme mémétique**.

RMQ: C'est actuellement l'approche dominante pour résoudre les benchmarks LSC avec n = 50, 60, 70, taux de pré-remplissage = 30% to 80%.

Métaheuristiques pour la résolution du COP

Deux grandes stratégies pour traiter les contraintes d'un COP.

- Stratégie 1 : on maintient la satisfaction de toutes les contraintes durant la recherche et explore uniquement des solutions réalisables (ou faisables).
- Stratégie 2 : on relaxe un certain nombre de contraintes et explore à la fois des solutions réalisables et non-réalisables (ou non-faisables).



Pourquoi relaxer des contraintes (Stratégie 2) ?

- Satisfaire toutes les contraintes peut être difficile (e.g., CSP).
- La relaxation de certaines contraintes peut faciliter (par franchissement de barrières) la découverte de (bonnes) solutions qui sont difficilement atteignables sans passer par des zones non-réalisables (cf. illustration).

RMQ

- La relaxation de contraintes entraîne des répercussions sur la **fonction d'évaluation**, le **voisinage** et le **croisement**.

Métaheuristiques pour le COP – fonction d'évaluation

Fonction d'évaluation étendue F (à minimiser) dans le cas de relaxation de contraintes.

$$\mathrm{F}(\mathrm{s}) = \mathrm{f}(s) + \alpha \mathrm{g}(s)$$

où

- f est la fonction de coût du problème (à minimiser) ;
- g est une fonction de pénalités pour les contraintes relaxées ;
- α est un paramètre qui contrôle l'importance accordée aux contraintes relaxées.

Le paramètre α permet de régler l'influence des pénalités sur F :

- Une petite valeur α pénalise moins la violation de contraintes et favorise davantage l'optimisation de f que la satisfaction de contraintes.

  ⇒ une trop faible valeur α induit un risque pour la recherche de rester trop souvent dans des zones non-réalisables.
- Une grande valeur α pénalise fortement la violation de contraintes et favorise davantage la satisfaction de contraintes que l'optimisation de f.

  ⇒ une trop grande valeur α peut empêcher le franchissement de barrières et diminue la chance de trouver des solutions réalisables de bonne qualité.

Métaheuristiques pour le COP – fonction d'évaluation

Comment déterminer une valeur appropriée pour α ?

Une stratégie consiste à faire évoluer *périodiquement* et *dynamiquement* le paramètre α (réglage auto-adaptatif).

- Si **aucune solution réalisable** n'a été visitée dans l'intervalle, la recherche est resté suffisamment longtemps dans des zones non-réalisables.

⇒ On augmente la valeur de α pour pénaliser un peu plus la violation de contraintes; ainsi on favorise davantage l'examen de solutions réalisables.

- Si **aucune solution non-réalisable** n'a été rencontrée dans l'intervalle, la recherche est restée suffisamment longtemps dans des zones réalisables.

⇒ On diminue la valeur de α pour moins pénaliser la violation de contraintes ; ainsi on favorise davantage l'examen de solutions non-réalisables.

Métaheuristiques pour le COP – voisinage et croisement

Le voisinage

1) Cas Stratégie 1 (recherche de solutions **réalisables**)

   Le mouvement doit générer uniquement des solutions réalisables ; si ce n'est pas le cas, chaque nouvelle solution doit être immédiatement réparée pour satisfaire les contraintes.

2) Cas Stratégie 2 (recherche de solutions **réalisables** et **non-réalisables**)

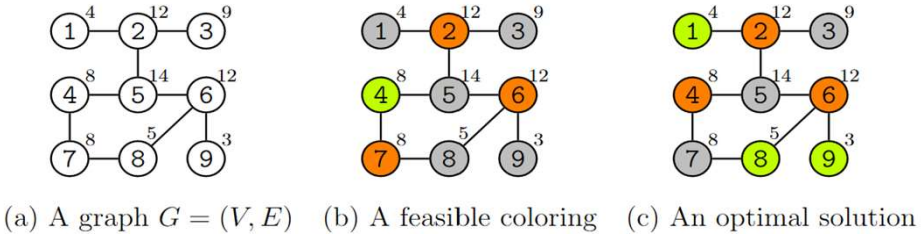
   Le mouvement doit générer des solutions qui respectent les contraintes non-relaxées ; si ce n'est pas le pas, la nouvelle solution doit être immédiatement réparée pour satisfaire les contraintes non-relaxées.

RMQ :

- Ces commentaires s'appliquent également au croisement pour une méthode à population.

Recherche local pour le COP – application à coloration pondérée

Coloration pondérée (weight vertex coloring) : Soit graphe $G = (V, E)$, et les poids associés aux sommets de V. Déterminer une coloration légale, i.e., *k* classes de couleurs $\{V_1, \dots, V_k\}$, pour minimiser le coût total **f** des *k* classes de couleurs, le coût d'une classe étant le plus grand poids de cette classe.



(a) A graph $G = (V, E)$ (b) A feasible coloring (c) An optimal solution

Colorations (b) et (c) ont 3 classes de couleurs: gris, orange et vert avec valeur de coût *f* :
(b): 14 (gris) + 12 (orange) + 8 (vert) = 34
(c): 14 (gris) + 12 (orange) + 5 (vert) = 31

L'espace de recherche Ω est composé de toutes les partitions des sommets en k classes de couleurs (i.e., contrainte de coloration relaxée). *k* peut diminuer ou augmenter contrairement au cas de k-COL.

La fonction d'évaluation étendue $F(s) = f(s) + \alpha g(s)$ où g(s) compte les conflits dans s, α est initié à 1, puis α++ ou α- - après chaque tour de recherche locale selon si une solution trouvée est réalisable.

Le voisinage « one-move » qui déplace un sommet entre 2 classes de couleurs quelconques.

Recherche local avec la métaheuristique Tabu Search comme pour la k-COL.

RMQ : Cette approche produit d'excellents résultats pour les benchmarks de la littérature.

Conclusion

Si vous avez un problème complexe sous contraintes que vous ne savez pas traiter autrement, alors tentez l'approche métaheuristique !

Quelques articles liés à l'exposé :

- **Résolution de contraintes (CSP) par recherche locale**
  - P. Galinier, J.K. Hao. A general approach for constraint solving by local search. *J. of Mathematical Modelling and Algorithms* 3(1): 73-88, 2004.
- **Recherche mémétique pour k-coloration & carré latin (CSP)**
  - P. Galinier, J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *J. of Combinatorial Optimization* 3(4): 379-397, 1999.
  - D.C. Porumbel, J.K. Hao, P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research* 37(10): 1822-1832, 2010.
  - L. Moalic, A. Gondran. Variations on memetic algorithms for graph coloring problems. *J. of Heuristics* 24(1): 1-24, 2018.
  - Y. Jin, J.K. Hao. Solving the Latin square completion problem by memetic graph coloring. *IEEE Transactions on Evolutionary Computation* 23(6): 1015-1028, 2019.
- **Exploration de solutions réalisables/non-réalisables (COP)**
  - F. Glover, J.K. Hao. The case for strategic oscillation. *Annals of Operations Research* 183(1): 163-173, 2011.
  - W. Sun, J.K. Hao, X. Lai, Q. Wu. Adaptive feasible and infeasible tabu search for weighted vertex coloring. *Information Sciences* 466: 203-219, 2018.

Merci !