Exploitation de structures de graphe en programmation par contraintes

Jean-Guillaume Fages

Sous la direction de :
Xavier Lorca et Nicolas Beldiceanu
TASC, EMNantes, CNRS, INRIA, LINA

29 juin 2015

Sommaire

Sujet de thèse

Contexte scientifique

Modéliser et résoudre des problèmes de décision

- La théorie des graphes (TDG)
- La programmation par contraintes (PPC)

Sujet de thèse

Contexte scientifique

Modéliser et résoudre des problèmes de décision

- La théorie des graphes (TDG)
- La programmation par contraintes (PPC)

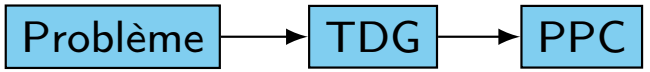
Problématique de la thèse

Comment combiner au mieux ces domaines pour mieux passer à l'échelle lors de la résolution de problèmes NP-difficiles ?

Démarche

Des contraintes pour les graphes

Résoudre des problèmes de graphe en PPC



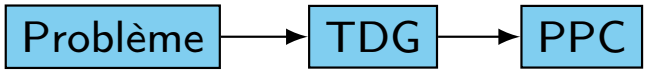
Démarche

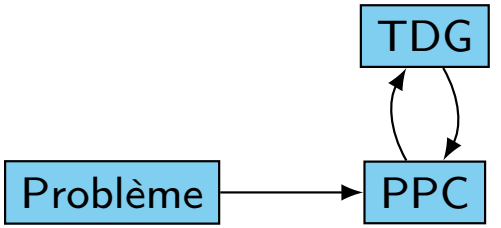
Des contraintes pour les graphes

Résoudre des problèmes de graphe en PPC

Des graphes pour les contraintes

Intégrer des raisonnements de la TDG dans les solveurs de PPC





Plan

La théorie des graphes

Un graphe

Un graphe $G = (V, E)$ est une abstraction représentant une relation binaire E sur un ensemble d'objets V.

- V : ensemble de nœuds ($|V| = n$)
- E : ensemble d'arêtes ($|E| = m$)

Un graphe est non-orienté ou orienté selon que la relation qu'il définit est symétrique ou non.

La théorie des graphes

Un graphe

Un graphe $G = (V, E)$ est une abstraction représentant une relation binaire E sur un ensemble d'objets V.

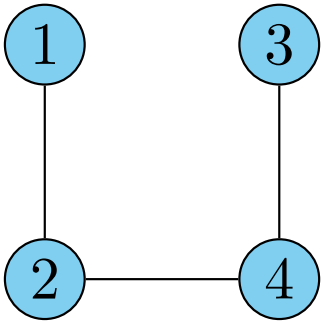
- V : ensemble de nœuds ($|V| = n$)
- E : ensemble d'arêtes ($|E| = m$)

Un graphe est non-orienté ou orienté selon que la relation qu'il définit est symétrique ou non.

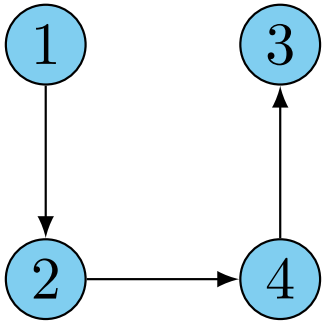
Exemples d'applications

- Conception de réseaux
- Analyse de programmes
- Biologie

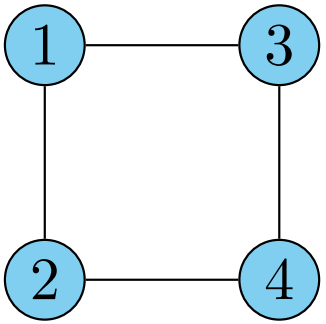
Exemples de graphes



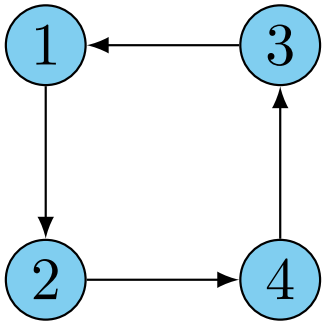
Chaîne



Chemin

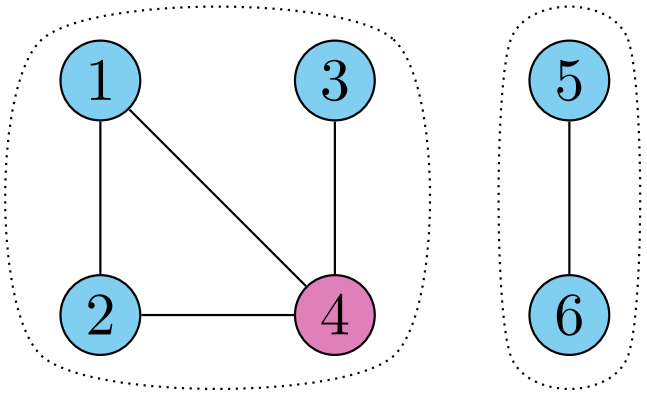


Cycle

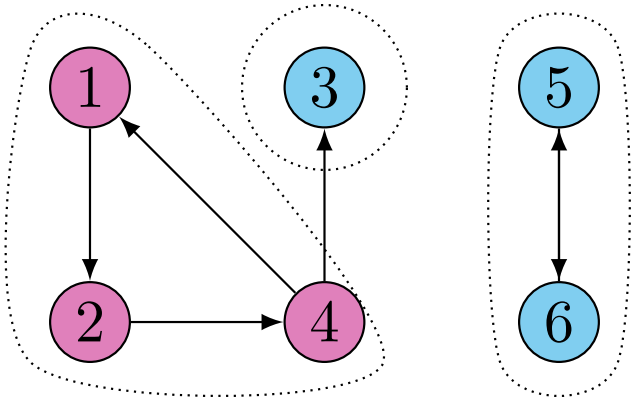


Circuit

Connexité

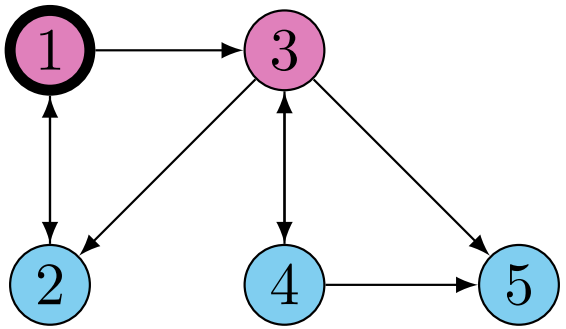


Composantes connexes et
points d'articulation

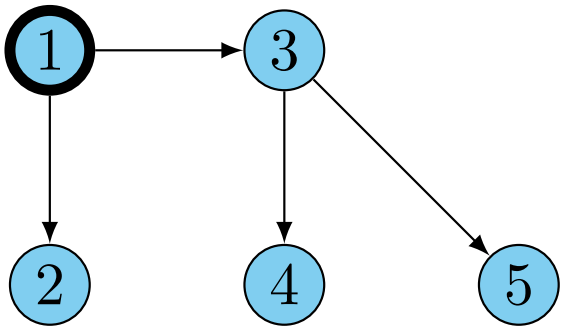


Composantes fortement connexes
et points forts d'articulation

Accessibilité



Graphe de flot **enraciné** en 1 et nœuds dominants



Arborescence **enracinée** en 1

Résumé

Intérêts de la TDG

- Montée en abstraction
- Générique
- Analyse structurelle
- Facilite le raisonnement

La programmation par contraintes

La programmation par contraintes (PPC)

Paradigme de programmation permettant de modéliser de manière déclarative et de résoudre des problèmes de satisfaction de contraintes (CSPs).

La programmation par contraintes

La programmation par contraintes (PPC)

Paradigme de programmation permettant de modéliser de manière déclarative et de résoudre des problèmes de satisfaction de contraintes (CSPs).

Exemples d'applications

- Ordonnancement des opérations de production
- Planification de personnel
- Vérification de programmes
- Configuration de produits

La programmation par contraintes

Problème de satisfaction de contraintes

- Un ensemble de variables $\mathcal{X} = \{x_1, \ldots, x_n\}$,
- Un ensemble de domaines $\mathcal{D} = \{d_1, \ldots, d_n\}$,
- Un ensemble de contraintes $\mathcal{C} = \{c_1, \ldots, c_m\}$ portant sur $\mathcal{X}$.

Une solution est une affectation, de chaque variable de $\mathcal{X}$ à une valeur de son domaine, satisfaisant l'ensemble des contraintes de $\mathcal{C}$

Algorithme de résolution

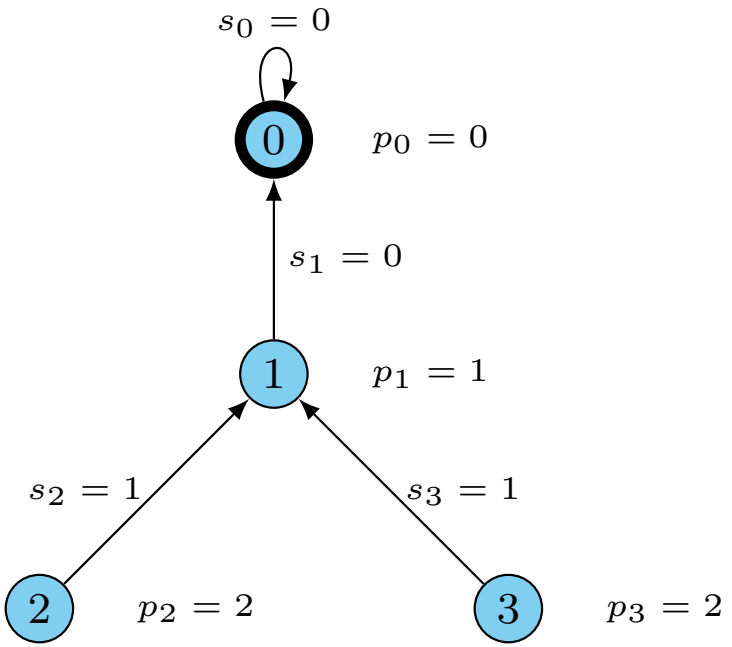
Alternance de propagation de contraintes (filtrage) et d'exploration.

Processus de résolution

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{ s_0 = 0 \wedge p_0 = 0$, // 0 est la racine
  $\bigwedge_{i \in [1,3]} s_i \neq i$, // autres nœuds non racine
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1$, // lien s/p
  $s_2 \neq 0 \wedge s_3 \neq 0$ // contrainte annexe
  $\}$

Successeur		Profondeur	
s_0	$\{0,1,2,3\}$	p_0	$\{0,1,2,3\}$
s_1	$\{0,1,2,3\}$	p_1	$\{0,1,2,3\}$
s_2	$\{0,1,2,3\}$	p_2	$\{0,1,2,3\}$
s_3	$\{0,1,2,3\}$	p_3	$\{0,1,2,3\}$



Une solution : anti-arborescence et affectation des variables

Processus de résolution : Filtrage

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{ \boldsymbol{s_0 = 0 \wedge p_0 = 0}$,
  $\bigwedge_{i \in [1,3]} s_i \neq i$,
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1$,
  $s_2 \neq 0 \wedge s_3 \neq 0$
  $\}$

Successeur		Profondeur	
s_0	$\{0, \cancel{1}, \cancel{2}, \cancel{3}\}$	p_0	$\{0, \cancel{1}, \cancel{2}, \cancel{3}\}$
s_1	$\{0, 1, 2, 3\}$	p_1	$\{0, 1, 2, 3\}$
s_2	$\{0, 1, 2, 3\}$	p_2	$\{0, 1, 2, 3\}$
s_3	$\{0, 1, 2, 3\}$	p_3	$\{0, 1, 2, 3\}$

Processus de résolution : Filtrage

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{ s_0 = 0 \wedge p_0 = 0,$
  $\bigwedge_{i \in [1,3]} s_i \neq i,$
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1,$
  $s_2 \neq 0 \wedge s_3 \neq 0$
  $\}$

Successeur		Profondeur	
s_0	$\{0\}$	p_0	$\{0\}$
s_1	$\{0, \not{1}, 2, 3\}$	p_1	$\{0, 1, 2, 3\}$
s_2	$\{0, 1, \not{2}, 3\}$	p_2	$\{0, 1, 2, 3\}$
s_3	$\{0, 1, 2, \not{3}\}$	p_3	$\{0, 1, 2, 3\}$

Processus de résolution : Filtrage

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{s_0 = 0 \wedge p_0 = 0,$
  $\bigwedge_{i \in [1,3]} s_i \neq i,$
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1,$
  $s_2 \neq 0 \wedge s_3 \neq 0,$
  $\}$

Successeur		Profondeur	
s_0	$\{0\}$	p_0	$\{0\}$
s_1	$\{0, 2, 3\}$	p_1	$\{0, 1, 2, 3\}$
s_2	$\{0, 1, 3\}$	p_2	$\{0, 1, 2, 3\}$
s_3	$\{0, 1, 2\}$	p_3	$\{0, 1, 2, 3\}$

Processus de résolution : Filtrage

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{s_0 = 0 \wedge p_0 = 0,$
  $\bigwedge_{i \in [1,3]} s_i \neq i,$
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1$
  $\boldsymbol{s_2 \neq 0 \wedge s_3 \neq 0},$
  $\}$

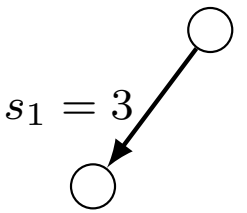
Successeur		Profondeur	
s_0	$\{0\}$	p_0	$\{0\}$
s_1	$\{0, 2, 3\}$	p_1	$\{0, 1, 2, 3\}$
s_2	$\{\cancel{\emptyset}, 1, 3\}$	p_2	$\{0, 1, 2, 3\}$
s_3	$\{\cancel{\emptyset}, 1, 2\}$	p_3	$\{0, 1, 2, 3\}$

Processus de résolution

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{ s_0 = 0 \wedge p_0 = 0,$
  $\bigwedge_{i \in [1,3]} s_i \neq i,$
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1,$
  $s_2 \neq 0 \wedge s_3 \neq 0$
  $\}$

Successeur		Profondeur	
s_0	$\{0\}$	p_0	$\{0\}$
s_1	$\{3\}$	p_1	$\{1, 2, 3\}$
s_2	$\{1, 3\}$	p_2	$\{0, 1, 2, 3\}$
s_3	$\{1, 2\}$	p_3	$\{0, 1, 2\}$



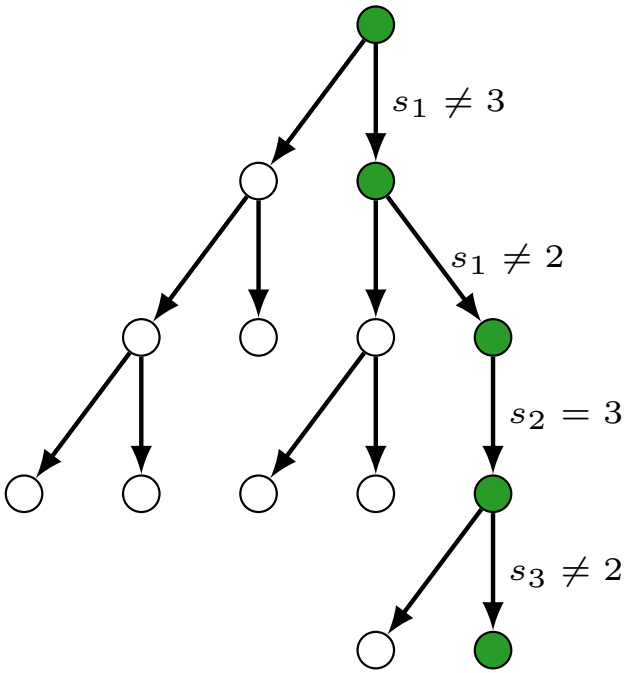
Arbre de recherche

Processus de résolution

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{ s_0 = 0 \wedge p_0 = 0,$
  $\bigwedge_{i \in [1,3]} s_i \neq i,$
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1,$
  $s_2 \neq 0 \wedge s_3 \neq 0$
  $\}$

Successeur		Profondeur	
s_0	$\{0\}$	p_0	$\{0\}$
s_1	$\{0\}$	p_1	$\{1\}$
s_2	$\{3\}$	p_2	$\{3\}$
s_3	$\{1\}$	p_3	$\{2\}$



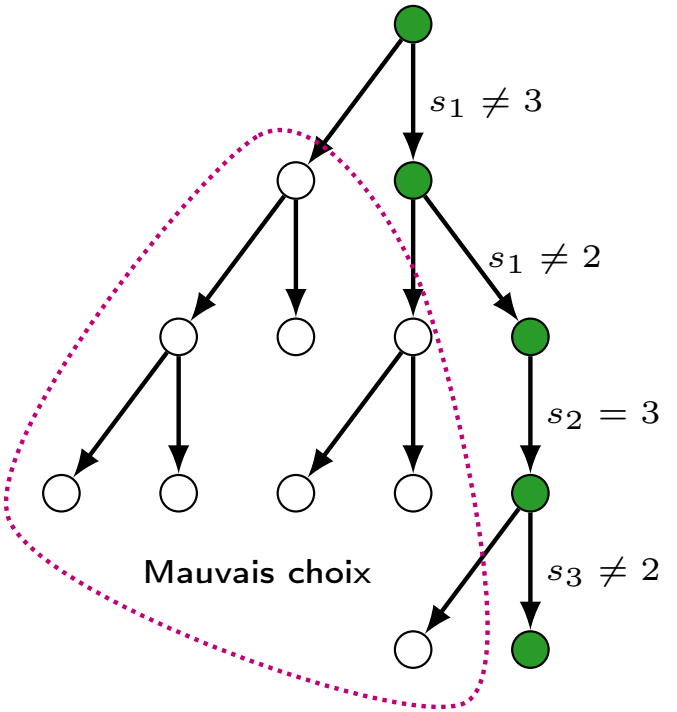
Arbre de recherche

Processus de résolution : Exploration

Exemple : modèle pour la recherche d'une anti-arborescence enracinée en 0

- $\mathcal{X} = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2, p_3\}$
- $\mathcal{D} = \{0, 1, 2, 3\}^8$
- $\mathcal{C} = \{ s_0 = 0 \wedge p_0 = 0,$
  $\bigwedge_{i \in [1,3]} s_i \neq i,$
  $\bigwedge_{i,j \in [0,3], i \neq j} s_i = j \Rightarrow p_i = p_j + 1,$
  $s_2 \neq 0 \wedge s_3 \neq 0$
  $\}$

Successeur		Profondeur	
s_0	$\{0\}$	p_0	$\{0\}$
s_1	$\{0\}$	p_1	$\{1\}$
s_2	$\{3\}$	p_2	$\{3\}$
s_3	$\{1\}$	p_3	$\{2\}$



Arbre de recherche

Résumé

Concepts

- Propagation = déductions
- Exploration = hypothèses (arbre de recherche)

Résumé

Concepts

- Propagation = déductions
- Exploration = hypothèses (arbre de recherche)

Axes de travail

- Comment filtrer plus ?
- Comment filtrer plus vite ?
- Comment prendre la meilleure décision ?
- Comment apprendre de ses erreurs ?

Résumé

Concepts

- Propagation = déductions
- Exploration = hypothèses (arbre de recherche)

Axes de travail

- Comment filtrer plus ?
- Comment filtrer plus vite ?
- Comment prendre la meilleure décision ?
- Comment apprendre de ses erreurs ?

Modélisation de graphes

Possible mais peu intuitive avec des variables entières

Plan

Exploitation de structures de graphe en programmation par contraintes

Résolution de problèmes de graphes

- Variables entières
- Variable de graphe

Résolution de problèmes de graphes en contraintes

Modèles à variables entières

- Voyageur de commerce [CL97 ; FLM02]
- Morphismes de graphes [Rég95 ; Sol10 ; Gay+14]
- Clique maximum [Rég03]
- Cycles [Bou99]

Résolution de problèmes de graphes en contraintes

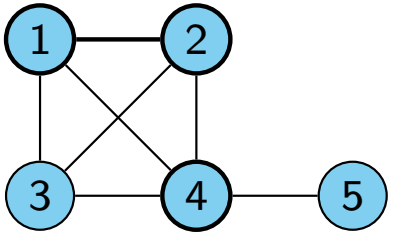
Introduction de variables et de contraintes sur les graphes

- Description de contraintes globales et filtrage [Bel00]
- Projet ROCOCO [LP+02]
- Tutoriel [Rég04]

Exploitation

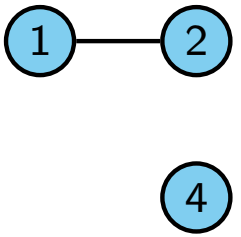
Problèmes d'arbres et de chemins [Doo06 ; Que06 ; Lor07]
Problèmes de couplages généralisés [Cym13]

Variable de graphe



$\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$

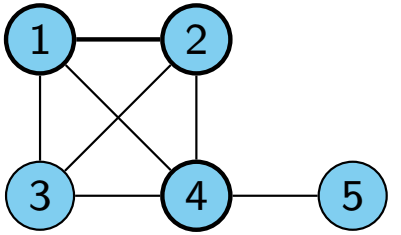
- Variable $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Domaine $[\underline{\mathcal{G}}, \overline{\mathcal{G}}]$
- Inclusion $\underline{\mathcal{G}} \subseteq \overline{\mathcal{G}}$
  (sous-graphe partiel)



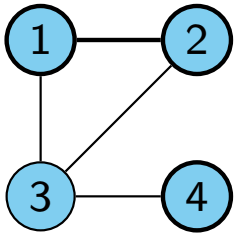
$\underline{\mathcal{G}} = (\underline{\mathcal{V}}, \underline{\mathcal{E}})$

Variable de graphe

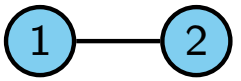
- Variable $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Domaine $[\underline{\mathcal{G}}, \overline{\mathcal{G}}]$
- Inclusion $\underline{\mathcal{G}} \subseteq \overline{\mathcal{G}}$
  (sous-graphe partiel)



$\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$



$\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$



$\underline{\mathcal{G}} = (\underline{\mathcal{V}}, \underline{\mathcal{E}})$

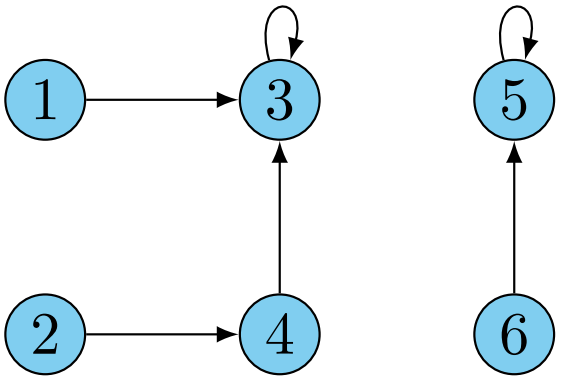
La contrainte Tree

Définition

La contrainte $\texttt{Tree}(\mathcal{G},N)$ est satisfaite si et seulement si $\mathcal{G}$ est un ensemble de N anti-arborescences.

Intérêt

- Modèle plus clair
- Filtrage complet [BFL05]



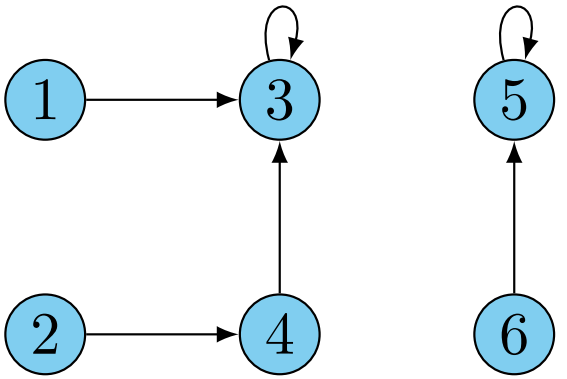
La contrainte Tree

Définition

La contrainte $\texttt{Tree}(\mathcal{G},N)$ est satisfaite si et seulement si $\mathcal{G}$ est un ensemble de N anti-arborescences.

Intérêt

- Modèle plus clair
- Filtrage complet [BFL05]



Exemples d'applications

Problèmes de phylogénie, réseaux, chemins et circuits.

La contrainte Tree

Complexité

- Filtrage sur le nombre d'arbres en $O(n+m)$ [BFL05]
- Filtrage structurel en $O(nm)$
  - Approche par *points forts d'articulation* [BFL05]
  - Approche par *nœuds dominants* et *fermeture transitive* [Que+06]

La contrainte Tree

Complexité

- Filtrage sur le nombre d'arbres en $O(n+m)$ [BFL05]
- Filtrage structurel en $O(nm)$
  - Approche par *points forts d'articulation* [BFL05]
  - Approche par *nœuds dominants* et *fermeture transitive* [Que+06]

Question ouverte

Existe-t-il un algorithme de filtrage complet qui soit linéaire ?

Amélioration de la contrainte Tree [FL11]

Démarche

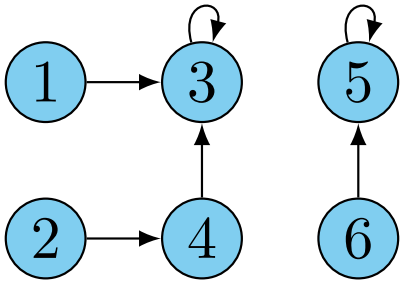
- Point de départ : [ILS10]
- Simplification : vers une contrainte d'arborescence

Définition

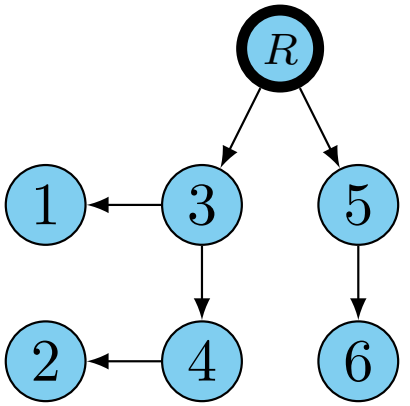
La contrainte $\texttt{Arbo}(\mathcal{G},R)$ est satisfaite si et seulement si $\mathcal{G}$ est une arborescence enracinée en R.

Transformation

$\texttt{Tree}(\mathcal{G},N) \Leftrightarrow \texttt{Arbo}(\mathcal{G}_2,R) \wedge \delta^+_{\mathcal{G}_2}(R) = N$



$\texttt{Tree}(\mathcal{G},N)$



$\texttt{Arbo}(\mathcal{G}_2,R)$

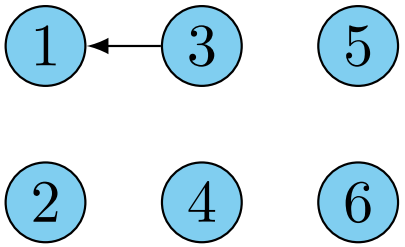
Amélioration de la contrainte Tree [FL11]

Nouvelle règle de filtrage

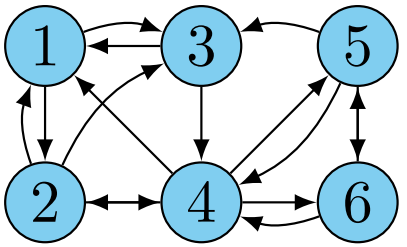
Un arc $(i, j) \in \overline{\mathcal{G}}$ n'appartient à aucune solution de $\texttt{Arbo}(\mathcal{G}, R)$ si et seulement si l'une des conditions suivantes est vraie :

- Il existe un nœud $k \neq i$ tel que $(k, j) \in \underline{\mathcal{G}}$
- j domine i dans le graphe de flot $\overline{\mathcal{G}}(R)$

Soit $R = 3$



$\underline{\mathcal{G}}$



$\overline{\mathcal{G}}$

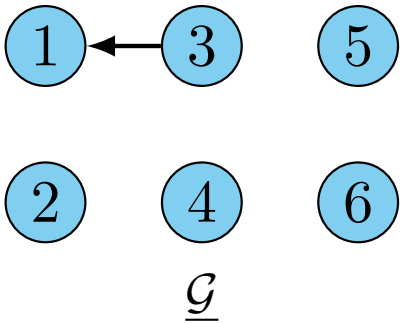
Amélioration de la contrainte Tree [FL11]

Nouvelle règle de filtrage

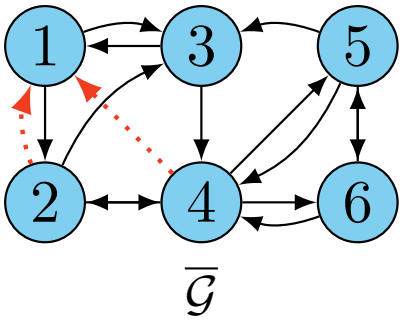
Un arc $(i, j) \in \overline{\mathcal{G}}$ n'appartient à aucune solution de $\texttt{Arbo}(\mathcal{G}, R)$ si et seulement si l'une des conditions suivantes est vraie :

- **Il existe un nœud $k \neq i$ tel que $(k, j) \in \underline{\mathcal{G}}$**
- j domine i dans le graphe de flot $\overline{\mathcal{G}}(R)$

Soit $R = 3$



$\underline{\mathcal{G}}$



$\overline{\mathcal{G}}$

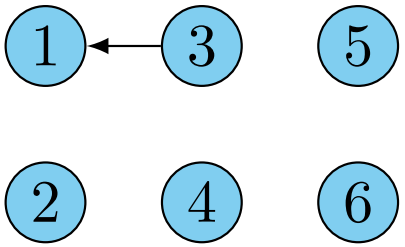
Amélioration de la contrainte Tree [FL11]

Nouvelle règle de filtrage

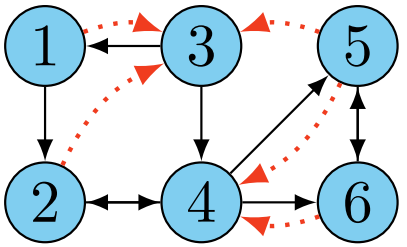
Un arc $(i, j) \in \overline{\mathcal{G}}$ n'appartient à aucune solution de $\texttt{Arbo}(\mathcal{G}, R)$ si et seulement si l'une des conditions suivantes est vraie :

- Il existe un nœud $k \neq i$ tel que $(k, j) \in \underline{\mathcal{G}}$
- **j domine i dans le graphe de flot $\overline{\mathcal{G}}(R)$**

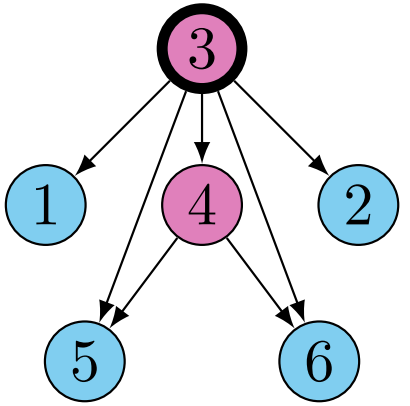
Soit $R = 3$



$\underline{\mathcal{G}}$



$\overline{\mathcal{G}}$



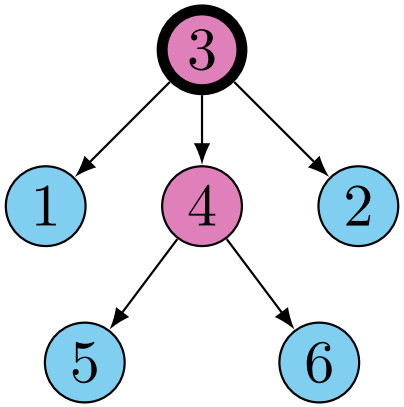
Dominance dans $\overline{\mathcal{G}}(3)$

Amélioration de la contrainte Tree [FL11]

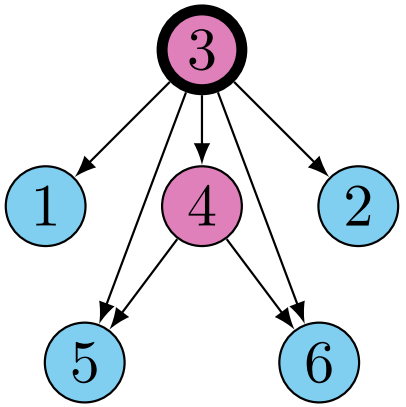
Complexité

Le filtrage complet est assuré en $O(n+m)$ [FL11]

- Arbre de dominance immédiate en $O(n+m)$ [Als+99]
- Numérotation *pre-ordre* et *post-ordre* des nœuds en $O(n)$
- Chaque requête de dominance en $O(1)$



Arbre de dominance immédiate



Graphe de dominance

Amélioration de la contrainte Tree [FL11]

Recouvrement d'un graphe par un ensemble d'anti-arborescences

Instances n	d^+	Décomposition [Lor11] temps (s)	nb. résolues	Tree [BFL05] temps (s)	nb. résolues	Tree [FL11] temps (s)	nb. résolues
50	5	1.1	80	0.5	**100**	**0.0**	**100**
	20	0.1	100	1.3	**100**	**0.0**	**100**
	50	0.1	100	1.2	**100**	**0.0**	**100**
150	5	2.6	60	4.5	**100**	**0.0**	**100**
	20	3.1	80	11.3	**100**	**0.0**	**100**
	50	0.6	87	25.3	**100**	**0.1**	**100**
	150	2.8	100	–	0	**0.3**	**100**
300	5	**0.1**	20	51.6	**100**	**0.1**	**100**
	20	0.4	47	–	0	**0.2**	**100**
	50	1.7	53	–	0	**0.5**	**100**
	300	17.7	77	–	0	**2.4**	**100**

- Génération de graphes aléatoires
- 100 instances par configuration
- 1 minute de résolution par instance

Amélioration de la contrainte Tree [FL11]

Recouvrement d'un graphe par un ensemble d'anti-arborescences

Instances n	d^+	Décomposition [Lor11] temps (s)	nb. résolues	Tree [BFL05] temps (s)	nb. résolues	Tree [FL11] temps (s)	nb. résolues
50	5	1.1	80	0.5	**100**	**0.0**	**100**
	20	0.1	100	1.3	**100**	**0.0**	**100**
	50	0.1	100	1.2	**100**	**0.0**	**100**
150	5	2.6	60	4.5	**100**	**0.0**	**100**
	20	3.1	80	11.3	**100**	**0.0**	**100**
	50	0.6	87	25.3	**100**	**0.1**	**100**
	150	2.8	100	–	0	**0.3**	**100**
300	5	**0.1**	20	51.6	**100**	**0.1**	**100**
	20	0.4	47	–	0	**0.2**	**100**
	50	1.7	53	–	0	**0.5**	**100**
	300	17.7	77	–	0	**2.4**	**100**

Instable

Rapide

Stable (GAC)

Lent

Stable (GAC)

Rapide

Utilisation de Arbo pour Path et Circuit

Définitions

- La contrainte Path($\mathcal{G}$,O,F) est satisfaite si et seulement si $\mathcal{G}$ est un chemin Hamiltonien allant de O à F, avec $O \neq F$.
- La contrainte Circuit($\mathcal{G}$) est satisfaite si et seulement si $\mathcal{G}$ est un circuit Hamiltonien.

Complexité

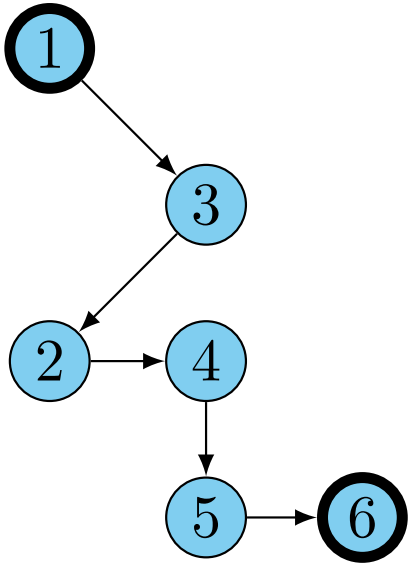
- Contraintes sur des problèmes NP-complets
- Pas d'algorithme de filtrage complet en temps polynomial
- Filtrage complet et polynomial possible sur des relaxations

Utilisation de Arbo pour Path

Reformulation

$\texttt{Path}(\mathcal{G}, O, F)$ reformulée en $\texttt{Arbo}(\mathcal{G}, O) \wedge \texttt{Arbo}(\mathcal{G}^{-1}, F)$

Accessibilité et élimination des sous-tours assurées



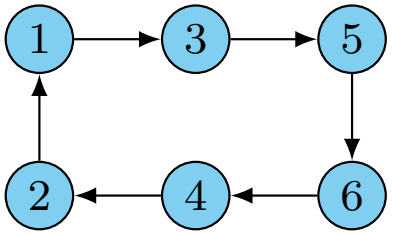
Utilisation de Arbo pour Circuit

Intuition

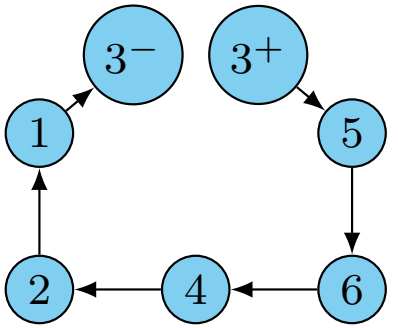
Un circuit est un chemin fermé

Reformulation de la contrainte Circuit

- Dupliquer un nœud i
  - i^- (arcs entrants)
  - i^+ (arcs sortants)
- On obtient le graphe $\mathcal{G}_i$
- Utiliser alors $\texttt{Path}(\mathcal{G}_i, i^+, i^-)$



$\texttt{Circuit}(\mathcal{G})$



$\texttt{Path}(\mathcal{G}_3, 3^+, 3^-)$

Forte connexité et élimination des sous-tours assurées

Utilisation de Arbo pour Circuit

Recherche d'un circuit Hamiltonien optimal

- $n = 60, m \approx 240$
- 500 instances [FS14]
- 10 minutes par instance

Approche	# résolues
Choco	93
Choco+Arbo	478
Chuffed [FS14]	487

Résultats

- Gain significatif sur le filtrage de Circuit
- Modèle compétitif avec les derniers résultats publiés [FS14]

Conclusion PPC $\Rightarrow$ TDG

Contributions

- Amélioration de `Tree` [FL11] : $O(nm) \Rightarrow O(n+m)$
- Utilisation pour `Path` et `Circuit`
- Autres algorithmes de filtrages [FL12]
- Etude empirique sur les heuristiques d'exploration [FLR14]

Perspectives

- Recherche d'un sous-arbre (mécanique)
- Autres problèmes de graphes
- Notion de robustesse

Plan

Plan

Conclusion

Synthèses

- La PPC traite bien de nombreux problèmes de graphe
  - Bon niveau d'abstraction
  - Plateforme d'intégration d'algorithmes de graphe
- La TDG facilite le raisonnement sur les contraintes
  - Analyser le modèle
  - Filtrer plus
  - Filtrer mieux
- Contributions théoriques et pratiques
- Outils disponibles

Publications

Journaux internationaux

- Artificial Intelligence, 2014 – Fages et Lapègue
- Constraints, 2014 – Fages, Lorca et Rousseau

Conférences internationales de rang A

- ECAI 2014 – Fages, Lorca et Petit
- CP 2013 – Fages et Lapègue, *Best student paper*
- CP 2011 – Fages et Lorca

Autres communications

TRICS'13, CP Solvers'13, JFPC'13, CPDP'12, ISMP'12, CoRR'12

Plan

RJCIA
29/06/2015

Choco

Solveur riche…

- Variables entières, continues, ensemblistes et de graphes
- Des centaines de contraintes (binaires et globales)
- Des dizaines de stratégies de recherche

…et flexible

- Création de stratégies de recherche ad-hoc
- Création de contraintes ad-hoc

COSLING

Choco

- Solveur Java, open source, licence BSD
- 15 ans d'expériences
  - Recherche
  - Industrie
- Hautes performances
  - MiniZinc 2014
  - MiniZinc 2013

Soutenu par l'INRIA et
l'Ecole des Mines de Nantes

Le Solveur de contraintes
le plus performant en Java

COSLING

Choco

Quelques applications :

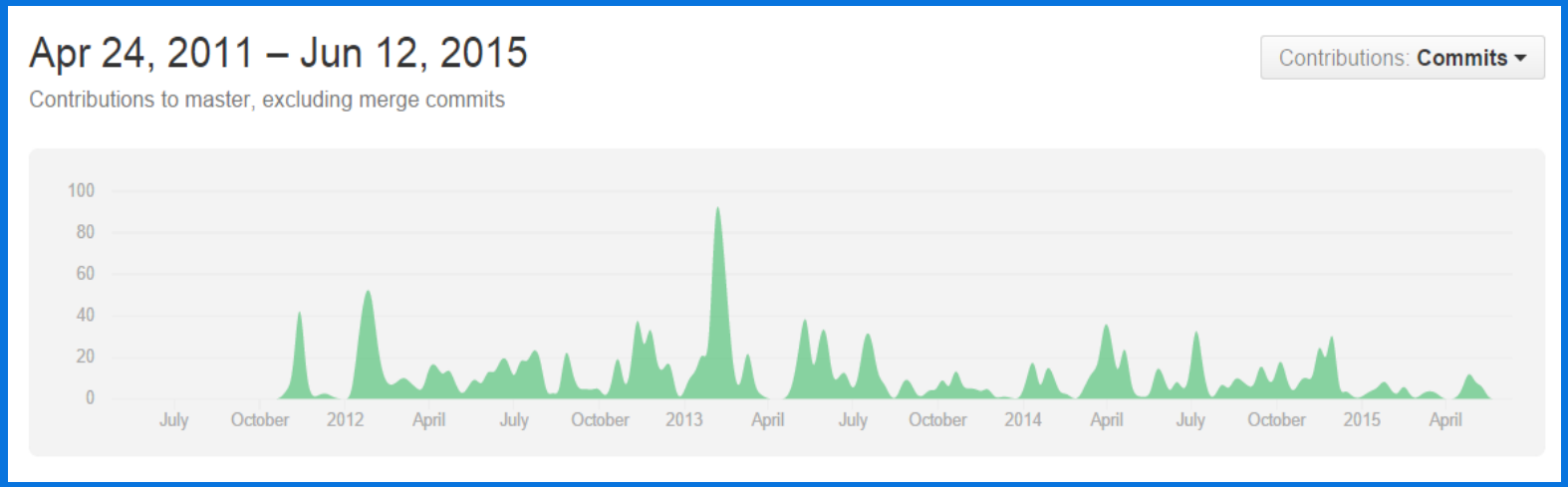
Industrie	Santé	Cloud
AIRBUS AN EADS COMPANY	medicalis	EasyVirt
DASSAULT AVIATION	BIOTRIAL Drug Evaluation and Pharmacology Research	HEDERA TECHNOLOGY

COSLING

Choco

Amélioration continue :

- Avancées en recherche
- Retours utilisateurs



COSLING

RJCIA
29/06/2015

Choco 4

Plus simple
Plus puissant

COSLING

RJCIA
29/06/2015

Plan

COSLING

CONSTRAINT-SOLVING

RJCIA
29/06/2015

COSLING

Jean-Guillaume FAGES

Ingénieur Mines-Télécom

Doctorat en Programmation par Contraintes – Prix AFIA

Contributeur majeur de Choco 3

Tanguy LAPEGUE

Ingénieur Mines-Télécom

Doctorat en Recherche Opérationnelle

Utilisation de Choco 3 en contexte industriel

COSLING

RJCIA
29/06/2015

COSLING S.A.S.

Création en novembre 2014

Activités

- Logiciels d'aide à la décision pour la performance industrielle
- Prestations en programmation par contraintes

Projets actuels

- Formations, conseil et support en programmation par contraintes
- Optimisation d'une chaîne de conditionnement
- Réparation des plannings du personnel

COSLING

RJCIA
29/06/2015

LabCom : EURODECISION / Ecole des Mines de Nantes

Optimisation de plannings de chauffeurs de bus

Particularité

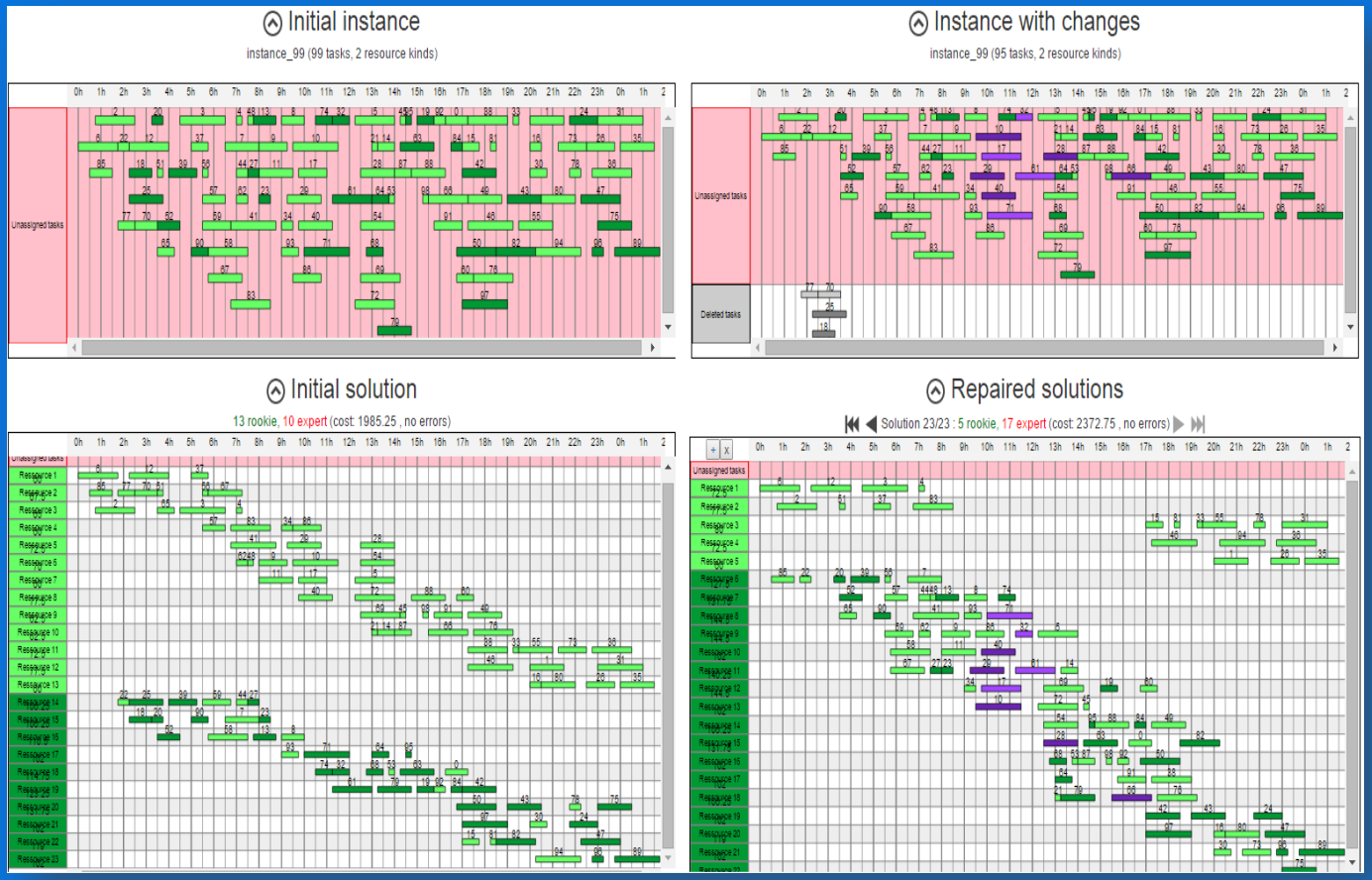
- Nombreuses contraintes de gestion
- Volonté de préserver au mieux un planning suite à un *aléa*

Rôles de COSLING

- Interface recherche/métier
- Développement d'un démonstrateur web
- Intégration des travaux de recherche

COSLING

Réparation d'un planning journalier



COSLING

RJCIA
29/06/2015

Recherche collaborative

GSCOP : *Constraints journal*

TASC : *CP'15*

CRIBA : *CP'15, Application track*

COSLING

Contactez-nous!

CONSTRAINT-SOLVING

COSLING

RJCIA
29/06/2015

Plan

Bilan

Faire une thèse, c'est explorer un arbre de recherche:

- On réfléchit (filtrage)
- On teste des hypothèses (exploration)
- On apprend de ses échecs… ou pas (explication)

Parfois il faut prendre son temps, parfois il faut foncer…

COSLING

Bilan

On ne sait jamais quelles idées vont vraiment marcher

Il faut mieux toutes les tester en parallèle

Et ça marche aussi pour les solveurs ;-)

COSLING

Bilan

Conseils pour les prochains:

- Soyez ouverts
- Travaillez en équipe
- Travaillez dur
- Prenez du plaisir!

COSLING

MERCI

COSLING

RJCIA
29/06/2015