REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE : MATHEMATIQUE ET INFORMATIQUE
DEPARTEMENT : INFORMATIQUE
N° :……..

DOMAINE : MATHEMATIQUE ET INFORMATIQUE
FILIERE : INFORMATIQUE
OPTION : SYSTEM D'INFORMATIONS ET GENIE LOGECIEL

Mémoire présenté pour l'obtention
Du diplôme de Master Académique

Par: Bousaid Mohamed El Amine

Intitulé

Programmations par contraintes de planifications d'horaire d'une ligue de football

Soutenu devant le jury composé de :

Ariout Youcef	Université de M'sila	Président
……	Université de M'sila	Rapporteur
……	Université de M'sila	Examinateur

Année universitaire : 2017 /2018

Remerciement

Grand merci à Allah, le tout puissant qui m'a donné la force et le courage d'arriver jusque-là.

Je tiens à remercier profondément mon encadreur : Monsieur " Ariouat Youcef "qui m'a encouragé à faire le maximum d'efforts dans ce travail ainsi que pour sa disponibilité.

Je remercie par la même occasion les membres de jury qui ont bien voulu accepter d'examiner et évaluer mon travail et participer à ma soutenance.

Je remercie aussi ma famille et tous les enseignants du département d'informatique pour leurs efforts afin de nous assurer la meilleure formation possible durant notre cycle d'étude.

Enfin, je remercie tous ceux qui ont contribué de près ou de loin à l'aboutissement de ce travail.

Dédicace

Louange a Allah de m'avoir donné la patience d'aller jusqu'au bout de mes rêves et le bonheur de lever mes mains vers le ciel et de dire

"Al Hamdoulillah"

Je dédie ce modeste travail à ceux qui mon aider dans ma vie, aux personnes généreuses, vertueuses qui m'ont été d'un grand secours, chacun a sa manière, qui ont sacrifiés leur temps pour mon bonheur et pour ma réussite, depuis l'école de mon enfance, et grâce a le éducation, mes parents, qui mon permis d'arriver à la fin de mon cycle d'étude

A l'esprit de mon père, et ainsi que ma mère qu'ils seront fière de moi.

Qu Allah les garde et les protèges ainsi que toute ma famille

Mes frères et mes sœurs,

A mes oncles, mes tantes, et leurs familles

A mes amis de la promotion

A tous ceux qui me sont chers.

Je dédie ce travail également a toutes les personnes chers a mon cœurs .Qu'elles trouvent ici l'expression de toute ma gratitude et mon amour.

SOMMAIRE

LISTE DES FIGURES

LISTE DES TABLES

INTRODUCTION GENERALE

Les problèmes de planification d'horaires concernent typiquement les sociétés et services exerçant une activité continue ou quasi-continue. Il en va ainsi par exemple des Compagnies aériennes, Sport, services portuaires, entreprises de transport, hôpitaux,… etc. La planification d'horaires vise à dimensionner la force à programmer l'utilisation de ressources sur un horizon allant de la journée à quelques mois, de manière à atteindre le meilleur équilibre entre qualité de service, coût et satisfaction sociale, l'objectif de cette mémoire consiste alors à définir et à mettre en œuvre une méthode et applications permettant de résoudre le problème de planification dans un cas concret : planning d'un tournoi sportif.

Les problèmes de planification d'horaires, de nature fortement combinatoire, appartiennent à la classe des problèmes dits NP-Complets, dont la résolution requière l'utilisation de techniques appropriées. Depuis Dantzig [1], de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO). Récemment, quelques méthodes ont vu le jour en intelligence artificielle (IA), ces méthodes peuvent être classées sommairement en deux grandes catégories : la première ; les méthodes exactes (complètes) qui garantissent la complétude de la résolution, parmi ces méthodes on trouve :

La programmation linéaire en nombres entiers [2], relaxation lagrangienne [3], …etc. La deuxième catégorie; les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité, parmi ces méthodes on trouve : la recherche tabou [4], le recuit simulé [5], les algorithmes génétiques Programmations par contraintes (PPC)[6],

Cette mémoire est structurée de la façon suivante :

Dans le chapitre 1 : on va présenter des notions générales sur les planifications des horaires et sa complexité au quotidien dans les entreprises et les associations sportives, on présente aussi la Problématique des Gestions des ligues et les techniques et les approches pour résoudre ce problème

Dans le chapitre 2 : on va présenter la programmation par contraintes en générale (historique, les domaines d'applications) et son principe, les solveurs existant est que l'on peut utiliser pour résoudre notre problème.

Dans le chapitre 3: Nous avons Présenté le fonctionnement et les caractéristiques de notre solveur nous avons présenté les étapes de la mise en œuvre de notre projet avec tous

les outils, les langages et les plateformes utilisés ainsi que la présentation avec l'explication du rôle de chaque outil. Nous avons illustré les interfaces graphiques avec une description textuelle.

CHAPITRE 1 : LE PROBLÈME DE PLANIFICATION D'UNE LIGUE

1. Introductions:

La première partie de ce chapitre met en scène la problématique de la planification des horaires dans un contexte général et sa complexité au quotidien dans les entreprises et Les associations sportives , dans la deuxième partie, nous entamons le problème de planification d'une ligue et nous allons expliquer pourquoi la planification d'une ligue est considérée comme un grand problème ?, enfin nous nous attaquons au problème de planification d'une ligue, les techniques possibles et les approches pour résoudre ce Problème.

2. Problématique de la planification d'horaires

Pour mieux cerner ce qui est la planification ainsi que la complexité inhérente à sa réalisation, nous commençons par donner les différentes définitions de la planification sous plusieurs formes.

2.1. Un processus complexe

La planification d'horaires de travail vise, pour un horizon de planification d'un jour à quelques mois, à dimensionner une force de travail et à optimiser l'utilisation des ressources, de façon à couvrir un besoin exprimé par une charge de travail prévisionnelle, tout en respectant un ensemble de contraintes précises, elle aboutit à des programmes définissant les horaires de travail et de repos de la force de travail [7].

Les contraintes évoquées ci-dessus sont relatives à la législation du travail ou à des accords spécifiques c'est-à-dire un ensemble de règles de planification liées aux habitudes de fonctionnement de l'entreprise ou à des conventions collectives et qui contribuent à la satisfaction du personnel (par exemple le nombre minimum de jours de travail successifs, l'absence de jours de repos isolés ou l'absence de journées de travail avec une coupure, ...).

2.2. Qu'est-ce que la planification ?

Afin de rester dans l'économie globale moderne, toute entreprise doit organiser et planifier le travail de ses salariés, ceci passe par la détermination des capacités des employés par le recensement des activités futures et des besoins en personnel, l'entreprise doit satisfaire ces derniers en affectant la bonne personne, à la bonne place au bon moment.

- *Définition opérationnelle de la planification :* La planification vise à affecter les ressources humaines pour chaque intervalle de temps sur un horizon donné, de telle sorte que les besoins par intervalle soient couverts et que les différentes contraintes soient satisfaites [7].
- *Définition mathématique de la planification :* La planification peut être définie sous la forme de fonction mathématique qui associe un salarié et un intervalle de temps à une affectation [7]. Exemple :

  fj(Salarié, Période) = Activité / Compétence.

  Sur la journée, cette fonction permet de déterminer les heurs de présences du salarié (ce qui définit sa vacation) et ses activités de la journée.

2.3. Qu'est-ce qu'un bon planning ?

Si un planning n'est pas toujours facile à obtenir, un bon planning l'est beaucoup moins. S'agissant des personnes, le coût d'un planning n'est pas la somme des coûts horaires des employés. En effet, la planification est rendue beaucoup plus complexe avec les contraintes et souhaits individuels. La satisfaction des salariés est un facteur important sur leur motivation et leur productivité.

Compte tenu des différents points de vue au sein d'une même entreprise ou des associations, des contraintes complexes et parfois contradictoires, créer un bon planning implique à la fois une négociation entre le planificateur et les différents employés concernés, et un calcul d'optimisation combinatoire.

2.4. Le contexte de la planification

2.4.1. Où faut-il planifier ?

Le problème du planning est central dans toutes les entreprises ou des associations sportives disposant de ressources humaines. En fait, le problème de planning apparaît systématiquement dans les situations suivantes :

- Si le travail doit être assuré pendant plus d'une journée, il faut prévoir la succession de plusieurs équipe sur le même stade dans la journée.
- Si l'équipe doit jouer un match par jour, un outil automatique devient indispensable lorsque le nombre des matches dépasse le maximaux pour gérer

la succession de plusieurs matches dans la semaine, ainsi que les absences Imprévues.

2.4.2. Planifier quoi ?

En se limitant à la planification des ressources humaines (sans prendre en considération les ressources matérielles), il faut décider si l'on planifie les horaires des matches ou les tâches effectuées par les équipes ou disponibilité de matériel par exemple des stades.

3. Planification d'une ligue

Chaque football ligue a besoin d'un calendrier, ce calendrier indique quelles équipes jouent les unes contre les autres à quel moment, un calendrier de championnat de football est établi avant le début de la compétition. De cette façon, les équipes en compétition savent exactement quand, où et contre qui elles doivent jouer, cela leur permet de planifier leur entraînement, leur préparation, leurs camps d'entraînement et d'autres activités en conséquence.

Faire un calendrier de la ligue de football n'est pas simple, généralement, de nombreuses contraintes doivent être prises en compte et ces contraintes sont différentes pour les ligues de football distinctes, de nombreuses parties prenantes sont impliquées et elles ont toutes leurs propres souhaits.

3.1. Le problème de planification d'une ligue

La définition du problème de planification d'une ligue sportive qui nous intéresse se présente à peu près comme suit:

« *trouver un calendrier optimal pour un ensemble d'équipes sur une période de temps qui réponde à un ensemble de contraintes particulières* ». Dans certains scénarios, il n'y a pas de calendrier réalisable, pour d'autres, il peut y en avoir beaucoup, et selon les contraintes, une peut être optimale.

L'établissement de l'horaire pour une association sportive est un travail qui doit être repris avant le début de chaque championnat en raison des changements qui apparaissent d'une année à l'autre, de nombreuses fédérations sportives sont confrontées à des problèmes de planification de matchs lors des tournois. Les règles qui régissent ces tournois ont autant de contraintes à prendre en compte avec les exigences des équipes,

3.2. Quelles sont les contraintes que l'on peut rencontrer lors de la planification ?

De nombreuses contraintes doivent être prises en compte et ces contraintes sont différentes pour les ligues de football distinctes, parmi ces contraintes on peut citer :

3.2.1. Contraintes de planification de base

1. Chaque équipe joue chacune des autres une fois au cours des 19 tours du tournoi.
2. Chaque équipe joue chaque tour à la maison ou à l'extérieur.
3. Chaque équipe joue au moins 9 tours, mais pas plus de 10, à domicile.

3.2.2. Contraintes de séquence de jeu à domicile et à l'extérieur

1. Chaque équipe joue au plus une séquence de deux tours consécutifs à domicile (stand à domicile). cela condition implique qu'aucune équipe ne joue plus de 2 tours consécutifs à la maison.
2. Chaque équipe joue au plus une séquence de deux rounds consécutifs (road trip). cette condition implique qu'aucune équipe ne joue plus de 2 tours consécutifs
3. "rounds d'ajustement" : si une équipe joue à la maison (à l'extérieur), il doit jouer loin (chez lui) dans le tour suivant.

3.2.3. Contrainte d'équilibre du jeu à domicile pour les matches contre des rivaux du groupe

Chaque équipe joue contre deux de ses adversaires à domicile et contre les deux autres adversaires.

3.2.4. Contraintes géographiques pour les séquences de parties en double

certaines contraintes ont été intégrées pour éviter des longs trajets routiers consécutifs

1. Quand une équipe du nord (sud) joue 2 matchs consécutifs à l'extérieur, aucun des deux ne sera dans le nord (sud).
2. Quand une équipe du nord (sud) joue 2 parties consécutives à l'extérieur, au moins un des jeux sera être dans le nord (sud).
3. Quand une équipe du centre joue 2 parties consécutives à l'extérieur, au moins une des parties sera en jeu. le centre.

3.2.5. Contraintes sur les matches à domicile et à l'extérieur pour les équipes "croisées"

Deux équipes sont qualifiées de "croisées" si leur lieu de résidence se situe dans la même région et si, pour des raisons opérationnelles (par exemple, ils partagent le même stade à la maison), pour des raisons de sécurité ou pour ne pas Pour quitter une région sans match pendant tout un week-end, ils alternent en jouant Matchs à domicile et à l'extérieur à chaque tour, lorsqu'une équipe d'une paire croisée joue à domicile, l'autre équipe joue et vice versa, inversement

3.2.6. Contraintes sur les classiques régionales

1. Un certain nombre de paires d'équipes de la même région ayant une rivalité historique sont définies comme un ensemble de classiques régionaux.

2. Les affrontements classiques régionaux ont lieu entre le 8ème et le 18ème round.

3.2.7. Contraintes liées au tourisme :

1. Un ensemble T est défini et contient des équipes situées dans des zones touristiques, où il est souhaitable qu'au moins un jeu attrayant (contre une équipe populaire) soit programmé pour les premiers tours du tournoi, pendant l'été.

2. Chaque équipe située dans une zone touristique joue à domicile contre au moins une des équipes populaires dans les 5 premiers tours.

3.2.8. Contraintes spéciales :

1. Certaines équipes n'ont pas leurs terrains de jeu prêts à temps pour le début du tournoi et devraient donc être programmées pour les matches à l'extérieur au premier tour. Ces équipes forment un mettre S.

2. Pas plus de 3 matchs entre les équipes du même groupe ont eu lieu au dernier tour.

3. Chaque équipe nord (sud) jouera au moins une fois à domicile contre une équipe du nord (sud), cette contrainte assurera que, lors du tournoi de clôture, les équipes nord et sud auront au moins un adversaire dans leurs groupes respectifs pour jouer un match à l'extérieur, cela pourrait aider à éviter les "mauvais" voyages sur route de deux jeux..

3.3. Les Approches réalisées pour résoudre ce problèmes (Etat de l'Art)

Cette section fournit une vue d'ensemble des méthodologies de solution les plus courantes utilisées pour la planification sportive, problèmes, je n'ai pas l'intention de fournir une liste exhaustive des méthodologies de solution, le plus pertinent et ceux largement utilisés sont introduits. Je n'ai pas non plus l'intention d'expliquer toutes les méthodologies en détail ; seuls les principes de base sont fournis et quelques exemples de références de programmation sportive, la structurée une grande partie du contenu de cette section repose sur [8].

3.3.1. Programmation en nombres entiers (Integer programming)

La programmation en nombres entiers traduit le problème de planification sportive en variables de décision, qui ne peuvent que prendre des valeurs entières, une fonction

objective et des contraintes, beaucoup de modèles utilisent généralement la définition de la définition de variable suivante :

X i, j, t = 1 si l'équipe joue à domicile contre l'équipe j au tour t,= 0 sinon pour les équipes i, j = 1, ..., n (avec i ≠ j) et arrondis t = 1, ..., r où n désigne le nombre d'équipes et r dénote le nombre de tours, les problèmes de programmation entiers peuvent être résolus en utilisant un logiciel d'optimisation tel que CPLEX, bien que les logiciels d'optimisation ne puissent pas résoudre le problème de manière raisonnable quantité de temps si le problème est très complexe, c'est aussi la raison pour laquelle l'hybridation avec d'autres les méthodes sont courantes dans la littérature : les méthodes de programmation par nombres entiers sont souvent utilisées comme solveurs de sous-problèmes dans les approches en solution impliquant plusieurs étapes.

3.3.2. Recherche heuristique et méta heuristiques

Les heuristiques sont utilisées lorsque d'autres méthodes ne permettent pas de résoudre le problème de quantité de temps de calcul, les heuristiques (ou algorithmes approximatifs) fournissent des solutions sous-optimales, mais ces solutions sont généralement proches des solutions optimales réelles et se retrouvent relativement rapidement. Différent les catégories d'heuristiques peuvent être différenciées :

- *Des heuristiques constructives* **:** permettent de créer des solutions réalisables à partir de zéro et sont souvent basées sur des solutions gourmandes les choix.
- *Les heuristiques de recherche locales :* sont basées sur la recherche de quartiers de solutions, successivement remplacer la solution actuelle par un meilleur dans son voisinage et se terminer quand non une meilleure solution peut être trouvée.
- *Les métas heuristiques :* sont des procédures de solution générales de haut niveau qui coordonnent des heuristiques simples études règles qui explorent l'espace de la solution pour trouver de bonnes solutions approximatives (souvent optimales),ils offrir différents mécanismes pour échapper aux solutions locales optimales (par exemple en permettant la détérioration et même des solutions irréalisables), contrairement aux heuristiques de recherche locale susmentionnées. Un méta heuristique se réfère à une stratégie maîtresse qui guide et modifie les autres procédures heuristiques pour produire des solutions au-delà de celles générées dans une quête d'optimalité locale, la personnalisation (ou l'instanciation) de certaines méta-heuristiques pour un problème donné produit une heuristique pour le dernier, voici quelques exemples de méta heuristiques :

a- Méthodes basées sur la population :

Algorithmes génétique : ils appartiennent à la classe des algorithmes évolutionnaires, qui sont des techniques de recherche pour des problèmes d'optimisation basés sur la "survie du plus apte "principe de génétique et sélection naturelle, des algorithmes génétiques peuvent être utilisés pour des problèmes binaires et continus

b-Méthodes basées sur une solution unique :

Recherche tabou : ceci est un algorithme de recherche local, ce qui signifie que les voisins d'une solution potentielle sont vérifiés dans l'espoir de trouver une solution améliorée. solution. pour éviter de rester coincé dans une région sous-optimale, la recherche tabou améliore les performances de la recherche locale de deux manières : l'algorithme accepte aggravation des mouvements et il comprend une liste tabu qui stocke les solutions qui sont considéré comme "tabu" et ne peut donc pas être sélectionné pour une nouvelle solution, recherche taboua déjà été appliqué à plusieurs reprises sur des problèmes de programmation sportive. [9], par exemple, utilisent cette méthode pour planifier Ligue de football belge. le lecteur est renvoyé au livre de [10] pour une explication plus détaillée.

c-Recuit simulé : c'est une technique qui guide la recherche locale, mieux bouge sont toujours acceptés, mais les pires mouvements peuvent également être acceptés en fonction de la fonction de probabilité d'acceptation.

Les hybridations combinant des principes de différent méta heuristique sont également populaires.

3.3.3. Approches de décomposition

Les stratégies de décomposition décomposent le problème de planification en sous-problèmes, qui sont ensuite résolus séquentiellement par des algorithmes exacts ou heuristiques, deux approches principales ont été étudiées dans la littérature pour différents paramètres de problème. Ils sont souvent utilisés en combinaison avec la programmation linéaire entière ou formulations de programmation par contraintes.

1."Premier programme, puis pause" [11], dans la première étape, les appariements sont déterminés pour chaque tour, c.-à-d. quelles équipes jouent les unes contre les autres dans un tour particulier, dans la deuxième étape, un le modèle de retour correspondant (HAP) est calculé pour ces couplages (souvent avec un minimum nombre de pauses).

2."Première pause, puis calendrier" [12], dans un premier temps, un chez soi réalisable modèle (souvent avec un nombre minimum de pauses) est déterminé., dans la deuxième étape, les paires pour le modèle sont fixes, dans ce cas, des espaces réservés sont parfois utilisés pour déterminer les paires et les équipes spécifiques sont affectées aux espaces réservés par la suite.

Bien qu'une technique pour résoudre des problèmes d'optimisation mathématique, plutôt qu'une décomposition stratégie, il est intéressant de mentionner la méthode de décomposition de Benders dans cette section, ceci est une solution méthode fréquemment utilisée dans les problèmes de planification sportive et elle est également utilisée pour résoudre la programmation stochastique problèmes, dans la décomposition de Benders, les variables du problème d'origine sont divisées en deux sous-ensembles de sorte qu'un problème principal de première étape soit résolu sur le premier ensemble de variables, et les valeurs pour le second un ensemble de variables est déterminé dans un sous-problème de deuxième étape pour une solution de premier niveau donnée, si le sous-problème détermine que les décisions fixes de première étape sont en fait irréalisables, alors des coupes sont générées et ajoutées au problème principal, qui est ensuite ré-résolu jusqu'à ce qu'aucune coupe ne puisse être généré.

Dans Rasmussen (2008), dans lequel est décrite la programmation de la ligue de football danoise, une logique basée La stratégie de décomposition de Benders est utilisée pour résoudre le problème d'optimisation, de cette façon, le problème est décomposé en différentes étapes. Une étape par exemple consiste à résoudre un problème de propriété intellectuelle, la méthode de résolution est une sorte de système de rétroaction (un algorithme itératif) qui recherche des horaires réalisables.

3.3.4. Programmation par contraintes

Pour appliquer cette méthodologie, le problème doit être modélisé par des ensembles de variables et de contraintes inters reliés, chaque variable à un domaine fini de valeurs d'instanciation possibles, l'objectif de la programmation par contraintes est de trouver des solutions réalisables plutôt que de trouver une solution optimale, un problème de programmation par contrainte peut même ne pas avoir de fonction objective, la programmation par contraintes dérive de nouvelles contraintes et fixe les valeurs des variables en manipulant les ensembles de contraintes et les domaines de variables originaux, de plus, les variables sont fixées à partir de conclusions logiques.

La programmation par contraintes a déjà été appliquée à plusieurs reprises sur des problèmes de programmation sportive, un exemple est l'article de Régin (2001), dans lequel l'efficacité de la programmation par contraintes pour minimiser le nombre de ruptures dans les problèmes de programmation sportive est affichée, pour une explication plus détaillée sur la programmation par contraintes, le lecteur est renvoyé au livre de [13].

4. Conclusion:

Dans ce chapitre on peut constater que la planification d'horaires est un processus complexe, Cette complexité s'accroît considérablement dans les modélisations des contraintes et prendre en compte les exigences des équipes et les parties prenantes.

Parmi tous les types de plannings, c'est sur les plannings sportifs que nous allons porter notre intérêt, et tout particulièrement sur les plannings des matches entre les équipes et nous avons présenté quelques approches existantes pour résoudre ce problème.

CHAPITRE 2 LA PROGRAMATION PAR CONTRAINTES

1. Introduction

La programmation par contraintes (PPC) est, au sens large, l'étude des systèmes de calcul basés sur les contraintes, ses origines mixtes se retrouvent en interfaces personne-machine dès les années soixante, en intelligence artificielle dans les années soixante-dix et en langages de programmation dans les années quatre-vingt, son application sérieuse et répandue au domaine de la recherche opérationnelle est plus récente; elle date des années quatre-vingt-dix, dans ce chapitre, nous traitons une partie seulement de la programmation par contraintes, celle qui s'intéresse à la résolution de problèmes combinatoires, nous présentons les concepts centraux regroupés selon les trois partie suivants: présentations de ppc et domaines, et les Principe et modélisations d'un ppc, on présente les solveur existants. la ppc pourra consulter quelques ouvrages (Marriot et Stuckey).

2. Présentation de la PPC

2.1. Historique

La programmation par contraintes (ppc) est une technique de résolution de problèmes qui vient de l'intelligence artificielle et qui est née dans les années 70 du siècle dernier, on peut dire que la PPC telle quelle existe s'est principalement inspirée :

- des travaux sur les [14], notamment ceux de J-L.Lauriere qui a proposé le système ALICE [15]
- de la programmation logique avec contraintes et principalement de Prolog[16] et des travaux de P. van Hentenryck qui a implémenté Alice en Prolog pour donner naissance au langage CHIP (Constraint Handling In Prolog).
- des problèmes de satisfaction de contraintes (CSP)[17],c'est avec le premier de ces trois domaines que la ppc est le plus proche, en effet, la programmation logique avec contrainte est basée ´ sur Prolog et la théorie des csp reste peu intéresse par la représentation du problème.

En programmation par contraintes (ppc), un problème est de fini à partir de variables et de contraintes, chaque variable est munie d'un domaine de finissant l'ensemble des valeurs possibles pour cette variable, une contrainte exprime une propriété qui doit être satisfaite par un ensemble de variables, en ppc, un problème est aussi vu comme une conjonction de sous-problèmes pour lesquels on dispose de méthodes efficaces de résolution, ces sous-problèmes peuvent être très simples comme $x < y$ ou complexe comme la recherche d'un flot Compatible, es sous-problèmes correspondent aux contraintes, la programmation par contraintes va utiliser pour chaque sous-problème une méthode de résolution spécifique à ce sous-problème, afin de supprimer les valeurs des domaines des variables impliquées dans le sous-problème qui, compte tenu des valeurs des autres domaines, ne peuvent appartenir a` aucune solution de ce sous problème, ce mécanisme est appelé filtrage. En procédant ainsi pour chaque sous problème, donc pour chaque contrainte, les domaines des variables vont se réduire. ´

Apres chaque modification du domaine d'une variable il est utile de réétudier l'ensemble des contraintes impliquant cette variable car cette modification peut conduire à de nouvelles déductions, autrement dit, la réduction du domaine d'une variable peut permettre de déduire que certaines valeurs d'autres variables n'appartiennent pas à une solution, ce mécanisme est appelé propagation, ensuite et afin de parvenir à une solution,

l'espace de recherche va être parcouru en essayant d'affecter successivement une valeur a toutes les variables, les mécanismes de filtrage et de propagation étant bien entendu relances après chaque essai puisqu'il y a modification de domaines parfois, une affectation peut duit ; le dernier choix d'affectation est alors remis en cause, il y a backtrack ou retour en arrière et une nouvelle affectation est tentée, la programmation par contraintes est donc basée sur trois principes : filtrage, propagation et recherche de solutions.

La programmation par contraintes est d'une grande souplesse puisque l'on peut Intervenir à plusieurs niveaux lors de la résolution d'un problème, notamment en :

- définissant de nouvelles contraintes, il faut donner alors un algorithme permettant de déterminer si la contrainte admet une solution pour un ensemble de domaines quelconque. Dans le meilleur des cas, l'algorithme de filtrage sera directement fourni (il s'agit alors de supprimer les valeurs qui n'appartiennent pas à une solution de la contrainte), ainsi n'importe quel algorithme de recherche opérationnelle pourra être utilisé en ppc, cependant, une utilisation efficace demandera une adaptation de l'algorithme en ppc. de nombreux outils de programmation par contraintes proposent des contraintes prédéfinies encapsulant de tels algorithmes.
- définissant des stratégies de recherche de solutions, il s'agit de définir des critères permettant de choisir la prochaine variable et la prochaine valeur qui sera affectée à cette variable, afin de mieux comprendre l'intérêt de cet aspect, on peut remarquer qu'un algorithme glouton correspond en fait à une stratégie qui n'est jamais remise en cause, d'une certaine manière on peut donc voir la ppc comme une généralisation des algorithmes gloutons pour les cas où l'on n'est pas certain de la validité de la stratégie gloutonne, la ppc est très souple puisqu'aucune hypothèse n'est faite ni sur le type des Contraintes utilisées ni sur les domaines des variables, par ailleurs, aucune solution ne peut être perdue puisque toutes les ´éventualités seront envisagées, c'est pourquoi on parle également d'algorithme énumératif, ans la suite, nous verrons que parmi les principes de la ppc que nous venons de présenter certains peuvent être relâchés, par exemple, il n'est pas n´nécessaire de supprimer toutes les valeurs qui ne satisfont pas une contrainte, on peut aussi chercher à obtenir plus de filtrage en étudiant a priori les conséquences de l'affectation de chaque valeur à chaque variable pour l'ensemble des contraintes.
-

2.2. Domaines d'application de la PPC :

La programmation par contrainte a pour ambition de résoudre n'importe quel type de problèmes combinatoires aussi elle a été utilisée pour une très grande variété D'applications réelles. Ainsi, on trouve un large éventail d'applications résolues à l'aide d'ILOG solver sur le site ILOG (www.ilog.fr) que nous reproduisons partiellement ici, nous avons regroupé par thèmes certaines applications trouvées sur ce site :

- *Planification et Gestion* : nous pouvons citer la planification de la logistique, de la distribution, du personnel, de l'affectation d'équipes, de missions, des techniciens d'installation et de maintenance, de missions satellitaires et de réseaux (dimensionnement et modélisation). ILOG Solver a également été utilisé pour la gestion delà chaine logistique et d'entrepôts, la configuration et diagnostic d'´équipements, l'ordonnancement de lignes de production, l'affectation de fréquences et de bande Passante et l'optimisation de charge.

- *Transport :* dans le domaine des transports, la ppc a fait ses preuves pour des applications telles que l'affectation d'´équipages, de comptoirs, de portes d'embarquement et de tapis a bagages, l'ordonnancement d'équipements, la gestion de flottes et la planification du trafic.

- *Commerce en ligne :* la ppc est utilisé pour résoudre des applications en ligne aussi variées que la gestion des commandes et des approvisionnements, le service de voyages, la gestion des crédits ou de conseils financiers.

- *Production industrielle :* chrysler a développé un système de planification de la production de ses véhicules intégrant les composants ILOG de PPC, l'application de planification gère la s´séquence des opérations de peinture des véhicules et améliore la productivité de 15 usines du groupe en Amérique du nord, au Mexique et en Europe. Ce système a déjà permis au producteur automobile de réduire ses couts de production de 500 000 dollars par an et par usine, soit une ´économie globale de7 à 9 millions de dollars par an.

- *Défense :* En Grande-Bretagne, la formation des 85 000 militaires de la BritishArmy est planifiée grâce a ILOG Solver.Dans chacun de ses domaines applicatifs, la PPC est utilisé par des acteurs de renom tels que SAP, Oracle, Daimler-Chrysler, Nissan, Peugeot-Citron (Production Industrielle) ; Siebel et Intershop (Commerce

Electronique) ; Sabre et SNCF(Transport) ; Airbus, Lockheed Martin et l'OTAN (aéronautique, Espace, défense);Deutsche Bank.

2.3. Principes de la PPC et problématique de la modélisation

2.3.1. Réseau de contraintes

Nous nous limiterons aux réseaux de contraintes à domaines finis, un réseau de contraintes fini N est défini par :

- un ensemble de variables X = {x1, . . . , xn},
- un ensemble de domaines D = {D(x1), . . . , D(xn)} ou D(xi) est l'ensemble fini des valeurs possibles pour la variable xi,
- un ensemble de contraintes C entre variables, une contrainte définit les combinaisons de valeurs des variables autorisées.

2.3.2. Algorithme de filtrage

Un algorithme de filtrage, encore appelé algorithme de réduction de domaines, est associe à chaque contrainte, son rôle est de supprimer des valeurs des domaines des variables de la contrainte pour lesquelles il n'est pas possible de satisfaire la Contrainte. Par exemple, pour la contrainte (x < y) avec D(x) = [10, 20], D(y) = [0,15], un algorithme de filtrage associe à cette contrainte pourra supprimer les valeurs de 15 à 20 de D(x) et les valeurs de 0 à 10 de D(y).Une des propriétés les plus intéressantes d'un algorithme de filtrage est la consistance d'arc. Un algorithme de filtrage associe à une contrainte réalise la consistance d'arc si il supprime toutes les valeurs des variables impliquées dans la contrainte qui ne sont pas consistantes avec la contrainte. Par exemple, pour la contrainte x+3 = y Avec les domaines D(x) = {1,3, 4, 5} et D(y) = {4,5, 8}, un algorithme de filtrage établissant la consistance d'arc modifiera les domaines pour obtenir D(x) = {1, 5} et D(y) = {4, 8}.

2.3.3. Mécanisme de propagation

D`es lors qu'un algorithme de filtrage associe a une contrainte modifie le domaine d'une variable, les conséquences de cette modification sont étudiées pour les autres contraintes impliquant cette variable. Autrement dit, les algorithmes de filtrage des Autres contraintes sont appelés afin de déduire éventuellement d'autres suppressions. On dit alors qu'une modification a été propagée. Ce m´mécanisme de propagation est répété jusqu' ce que plus aucune modification n'apparaisse, comme

les domaines Sont finis et comme un algorithme de filtrage est appelé au plus une fois pour chaque modification ce processus se termine nécessairement.

L'idée sous-jacente à ce mécanisme est d'essayer d'obtenir des déductions globales. En effet, on espéré que la conjonction des déductions obtenues pour chaque contrainte prise indépendamment conduira à un enchaînement de déductions c'est-à-dire que cette conjonction est plus forte que l'union des déductions obtenues Indépendamment les unes des autres.

2.3.4. Mécanisme de recherche de solutions

Historiquement, le modèle théorique de la PPC, et plus particulièrement des CSP (Contraint Satisfaction Problème), avait pour but de résoudre des problèmes de satisfaction. Aussi, une solution est considérée comme étant une instanciation des variables qui satisfont toutes les contraintes. Lors de la résolution de problèmes d'optimisation, on distinguera deux types de "solutions": les solutions du problème de satis faisabilité sous-jacent, c'est-à-dire celles qui ne tiennent pas compte du coût, et les solutions optimales, c'est-à-dire celles qui minimisent (ou maximisent) la fonction de coût. Le mécanisme de recherche de solutions a pour but de trouver une solution, éventuellement optimale. Il met en œuvre les différents moyens qui vont permettre au solveur d'atteindre des solutions. Parmi ces moyens nous pouvons citer :

- les stratégies de choix de variables et de valeurs, elles définissent les critères qui vont permettre de déterminer la prochaine variable qui sera instanciée ainsi que la valeur qui lui sera affectée.
- les méthodes de décomposition, lorsque le problème est trop gros, il est souvent nécessairement de le décomposer en plusieurs parties, puis de résoudre ces parties de façon plus ou moins indépendante et enfin de les recombiner.
- les améliorations itératives, il est souvent illusoire de vouloir trouver et prouver l'optimalité d'un problème de grande taille. Aussi, bien souvent, on recherche quelques "bonnes" solutions puis on essaie de les améliorer à l'aide de techniques d'améliorations locales.

2.4. Modélisation :

Dans cette section, nous présentons la problématique de la modélisation, puis, nous nous attardons sur le concept de contraintes globales qui est majeur en PPC, car ces contraintes contiennent beaucoup plus d'informations. Enfin, nous donnons

un exemple de modélisation efficace, pour résoudre un problème à l'aide d'un solveur, un utilisateur doit définir le Réseau de contraintes qu'il considère ainsi que les méthodes de résolutions et les stratégies de choix de variables et de valeurs, la modélisation d'un problème se fait donc par l'identification de sous-problèmes aisés à résoudre qui vont correspondre aux contraintes choisies. Trois types de contraintes sont à la disposition de l'utilisateur :

- contraintes prédéfinies du solveur (ex. contraintes arithmétiques, de cardinalité, ...)
- contraintes données en extension, autrement dit par l'ensemble des combinaisons autorisées ou bien interdites
- les contraintes correspondant à des combinaisons de contraintes utilisant les opérateurs logiques ET, OU, XOR, NOT. Elles sont parfois appelées méta-contraintes.

En outre, l'utilisateur peut d´définir ses propres contraintes, en établissant la sémantique de la contrainte, ainsi que l'algorithme de filtrage associé, une contrainte peut également ˆêtre vue comme la recherche de conditions nécessaires vérifiées par toute solution. L'algorithme de filtrage associé à la contrainte se charge alors de supprimer du domaine des variables les éléments qui ne satisfont pas la condition. L'une des difficultés majeures de la PPC et notamment de la modélisation est l'identification des contraintes, pour que la résolution ait une chance d'ˆêtre efficace, deux conditions doivent généralement être remplies :

(i) les contraintes doivent être fortes afin d'engendrer des modifications des domaines des variables.

(ii) les modifications dues à un filtrage doivent pouvoir être utilisées par les autres contraintes.

Ces deux points méritent d'être un peu détaillés.

(i) Le risque de la modélisation est de représenter le problème initial soit comme un ensemble de sous-problèmes très locaux, c'est-à-dire que le problème initial est trop décomposé, soit à l'aide de sous-problèmes correspondant à des relaxations trop fortes du problème réel, dans le premier cas, les contraintes expriment des conditions très locales et donc trop indépendantes du problème initial. Dans le dernier cas, les contraintes correspondent à des conditions globales mais trop éloignées du problème. Dans les deux cas, les déductions se produiront beaucoup trop tardivement pendant la recherche, alors

que l'idéal est que les algorithmes de Filtrage associés aux contraintes déduisent rapidement des incohérences afin d'éviter d'étudier des parties importantes de l'espace de recherche.

(ii) Certaines contraintes sont incapables de tirer partie de la déduction faite par d'autres contraintes. Ainsi, après initialisation, la contrainte ($x < y$) ne peut déduire quelque chose que lors des modifications des bornes de x ou de y. Cela Signifie que si le filtrage associé `a une contrainte supprime une valeur de x qui n'est ni la valeur minimale de D(x), ni la valeur maximale de D(y) alors le filtrage associe a ($x < y$) ne fera aucune déduction.

2.4.1. Contraintes globales

Comme la PPC est basée sur des algorithmes de filtrage, il est particulièrement Important de d´définir des algorithmes efficaces et puissants, aussi, ce thème a attiré l'attention de nombreux chercheurs, qui ont d´découvert de nombreux algorithmes, néanmoins, de nombreuses études sur la consistance d'arc se sont limitées aux contraintes binaires définies en extension, c'est-à-dire par la liste des combinaisons de valeurs autorisées. Cette limitation peut être justifiée par le fait que n'importe quelle contrainte peut toujours ˆêtre définie en extension et par le fait que tout réseau de contraintes non binaires peut être transformé en un réseau équivalent n'impliquant que des contraintes binaires et un certain nombre de variables additionnelles [Rossi et al. 1990], cependant, en pratique, cette approche a de nombreux défauts :

- il est souvent inconcevable de transformer une contrainte non binaire en un ensemble de contraintes binaires à cause du coût de traitement d'une telle opération et de la m´mémoire requise (particulièrement pour les contraintes dites "Non représentables" cf. [18]).

- la structure des contraintes n'est absolument pas exploitée. cela empêche le développement d'algorithmes de filtrage efficaces dédiés aux contraintes. De plus, de nombreuses contraintes non binaires perdent totalement leurs structures lorsqu'elles sont représentées par un ensemble de contraintes binaires. Cela conduit, par exemple, a moins de filtrage de la part des algorithmes de filtrage par consistance d'arc associés à ces contraintes, en effet, deux réseaux de Contraintes ´équivalents en termes de solutions, n'auront pas nécessairement les mêmes domaines après fermeture par consistance d'arc de chaque contrainte. l'intérêt de l'utilisation de la structure des contraintes peut être mis en évidence l'aide d'un exemple simple : la contrainte $x \leq y$. soient min(D) et max(D)

respectivement la valeur minimum et la valeur maximum d'un domaine D, il est ´évident que toutes les valeurs de x et de y de l'intervalle [min (D(x)), max (D(y)]Sont consistantes avec la contrainte. Cela signifie que la consistance d'arc peut être Facilement et efficacement réalisée en supprimant toutes les valeurs qui ne sont pas dans l'intervalle ci-dessus. Par ailleurs, l'utilisation de la structure des contraintes est Souvent la seule manière d'éviter les problèmes de consommation mémoire liés aux contraintes non binaires. En fait, cette approche évite de représenter explicitement toutes les combinaisons de valeurs autorisées par la contrainte.

En conséquence, les chercheurs intéressés par la résolution d'applications réelles avec la PPC, et particulièrement ceux qui d´enveloppent des langages de PPC (comme PROLOG), ont écrit des algorithmes de filtrage spécifiques aux contraintes simples les plus communes (comme =, 6=, <, ≤, ...) ainsi que des cadres formels généraux Permettant d'exploiter efficacement certaines structures des contraintes binaires. Les chercheurs ont alors été confrontés deux nouveaux problèmes : le manque d'expressivité de ces contraintes simples et la faiblesse des réductions de domaines entrainés par les algorithmes de filtrage Associés à ces contraintes. Intéressons-nous tout d'abord à l'expressivité. Il est, en effet, beaucoup plus agréable pour modéliser un problème en PPC d'avoir à sa disposition des contraintes correspondant à un ensemble de contraintes simples. Ces contraintes encapsulant un ensemble d'autres contraintes sont appelées contraintes globales formellement, On a :

Définition 1 Soit C = {C1, C2, .., Cn} un ensemble de contraintes, la contrainte CG égale `a la conjonction de toutes les contraintes de C : CG = ∧{C1, C2, ..Cn} est une contrainte globale, l'ensemble de tuples de C est égala l'ensemble de solutions de(∪C∈CX(C), DX(C), {C1, C2, .., Cn}).Par exemple, une contrainte alldiff définie sur un ensemble X de variables impose que les valeurs prises par ces variables soient deux à deux différentes, il est beaucoup Plus simple de définir une seule contrainte alldiff(X), plutôt que de définir une contrainte de différence entre chaque paire de variables de X .De plus, ces nouvelles contraintes peuvent être associées à des algorithmes de filtrage beaucoup plus puissants parce qu'elles peuvent prendre en compte simultanément la présence de plusieurs contraintes simples afin de réduire plus le domaine des variables. Nous pouvons mettre en évidence cet aspect avec un exemple plus réaliste qui implique des contraintes globales de cardinalité (GCC).

Une GCC est définie par un ensemble de variables X = {x1, ..., xp} qui prennent leurs valeurs dans un ensemble V = {v1, ..., vd}, elle contraint le nombre de fois où chaque

valeur vi ∈ V est affecté à une variable de X a appartenir à un intervalle [li, ui], les GCC apparaissent dans de nombreux problèmes réels considérons

Exemple :

	Mo	Tu	We	Th
Peter	D	B	O	M
Paul	D	N	M	N
Mary	N	O	D	D

A = {M, D, N, B, O}, P = {Peter, Paul, mary, ...}

W = {Mo, Tu, We, Th, ...}

M : morning, D : day, N : night B : backup, O : day-off

Figure 2. 1: Un problème d'emploi du temps

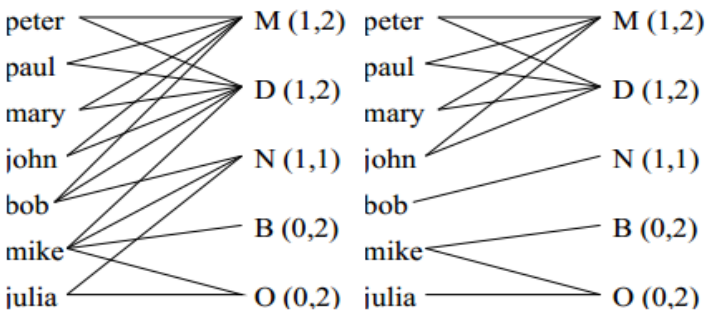

Figure 2. 2: Un exemple de contrainte globale de cardinalité

Un exemple dérive d'un problème réel et présente par [19] (cf. Figure 2.1), le but est de définir l'emploi du temps de managers d'un centre d'assistance comportant 5 activités, représentes par l'ensemble A, 7 personnes, représentées par l'ensemble P pour une période de 7 jours, représentée par l'ensemble W, chaque jour, une personne doit effectuer une des activités de l'ensemble A. Le but est de définir une matrice d'affectation qui satisfait les contraintes générales et locales suivantes :

- Les contraintes générales restreignent les affectations, pour chaque jour, chaque activité doit être affectée un certain nombre de fois compris entre un minimum et un maximum donnés. Pour toute période de 7 jours, une personne doit

effectuer un certain nombre de fois chaque activité, aussi, pour chaque ligne et pour chaque colonne de la matrice, une contrainte de cardinalité globale est définie.

- Les contraintes locales indiquent principalement des incompatibilités entre deux jours consécutifs, par exemple, on ne peut pas travailler le matin après avoir travaillé la nuit précédente.

Chaque contrainte générale peut être représentée par autant de contraintes min/max qu'il y a d'activités. Ces contraintes min/max peuvent être facilement manipulées grâce, par exemple, aux opérateurs atmost/atleast proposés par[18]. De tels opérateurs sont implémentes sous forme d'algorithmes de filtrage locaux. Or, il a ´été remarque dans [19] :le problème est qu'une résolution efficace de problèmes d'emploi du temps requiert un calcul global pour l'ensemble des contraintes min/max, et non pas une implémentation efficace de chacune d'elles séparément", c'est pourquoi, cette Manière de procéder n'est pas satisfaisante. Aussi, les contraintes globales de cardinalité associées à des algorithmes de filtrage efficaces (comme ceux réalisant la consistance d'arc) sont nécessaires. Afin de montrer les différences entre un filtrage global et un ensemble de filtrage locaux, nous considérons une GCC associée à une journée (voir figure 2.2), cette contrainte peut être représentée par un graphe biparti (graphe de gauche dans la Figure 2.2), l'ensemble gauche correspond à l'ensemble des personnes et l'ensemble droit à l'ensemble des activités, il existe une arête entre une personne et une activité lorsque la personne peut être affectée à l'activité. Pour chaque activité les nombres entre parenthèses indiquent le nombre minimum et maximum de personnes qui peuvent être affectées à l'activité, par exemple, john peut travailler le matin et la journée, mais pas la nuit ; au moins un manager doit travailler le matin, et au plus deux managers peuvent travailler le matin, nous rappelons que chaque personne doit être affectée à une et une seule activité, modéliser le problème avec un ensemble de contraintes atmost/atleast ne conduit aucune suppression de valeur. En effet, une contrainte atleast(X, #time, val) est une contrainte locale si une telle contrainte est considérée individuellement alors la valeur val ne peut pas être supprimée d'un domaine tant qu'elle appartient plus de#time fois aux domaines des variables de X. Un algorithme de filtrage par consistance d'arc pour cette contrainte affectera une variable x aval si et seulement si Il ne reste plus exactement que #time variables dont le domaine contient val. De Façon similaire, si une contrainte atmost(X, #time, val) est considérée individuellement alors la valeur val ne peut pas ˆêtre supprimée d'un domaine tant que

#time variables n'ont pas été affectées à cette valeur, un algorithme de filtrage par consistance d'arc pour cette contrainte supprimera val du domaine d'une variable x si et seulement si #time variables différentes de x sont affectées à val, on remarquera qu'aucun de ces cas ne se produit pour l'exemple considère or, avec une étude particulière de la contrainte on peut déduire certaines choses.Peter, Paul, Mary, et John peuvent travailler uniquement le matin ou la journée, déplus, le matin et la journée ne peuvent être affectés au plus qu'à quatre personnes, donc, aucune autre personne (i.e. Bob, Mike, ou Julia) ne peuvent effectuer les Activités M et D, aussi, nous pouvons supprimer les arêtes entre Bob, Mike, Julia Et D, M, autrement dit éliminer les valeurs D et M des variables correspondant aux Personnes Bob, Mike et Julia. Maintenant, il n'y a plus qu'une seule possibilité pour Bob : N, qui ne peut être affecté qu'au plus une fois. C'est pourquoi, nous pouvons supprimer les arrêtés {mike,N} et {julia,N}, ce raisonnement conduit au graphe de droite de la Figure 2.2. Il correspond à la réalisation de la consistance d'arc pour la Contrainte globale de cardinalité, le filtrage est une propriété locale, si on décompose une contrainte, on obtient alors un ensemble de filtrages plus faibles car moins informé. Nous pouvons formellement mettre en évidence ces différences entre filtrages par la propriété suivante :

Propriété : 1 la réalisations de la consistance d'arc pour une contrainte C = ∧ {C1, C2, ..,Cn} est plus forte (autrement dit ne peut pas supprimer moins de Valeurs) que la réalisation de la consistance d'arc du réseau(∪C∈CX(C), DX(C), {C1, C2, .., Cn}).preuve : d'après la définition 1 l'ensemble des tuples de C = ∧{C1, C2, .., Cn} correspond `a l'ensemble des solutions du réseau (∪C∈CX(C), DX(C), {C1, C2, .., Cn}). Donc, la réalisations de la consistance d'arc pour ∧{C1, C2, .., Cn} supprime toutes les valeurs qui n'appartiennent pas `a une solution de (∪C∈CX(C), DX(C), {C1, C2, .., Cn}) ce qui est plus Fort que réaliser la consistance d'arc sur ce réseau, aussi la consistance d'arc pour une contrainte globale est une propriété forte, la proposition suivante montre cette force en exhibant un réseau de contrainte pour Lequel la consistance d'arc est équivalente à celle d'une contrainte alldiff. Proposition 1 la consistance d'arc pour C =alldiff(X) correspond `a la consistance d'arc pour le réseau de contrainte ayant un nombre exponentiel de contraintes défini par :

$\forall A \subseteq X : |D(A)| = |A| \Rightarrow D(X - A)$ est réduit `a $D(X) - D(A)$

Preuve : voir [Régin, 1995].Enfin, remarquons qu'il est également possible de définir des algorithmes de filtrage simple pour les contraintes globales juste en prenant en compte la

présence simultanée de contraintes, considérons, par exemple un ensemble de 5 variables : X= {x1, x2, x3, x4, x5}

Dont les domaines sont D(x1) = 0, D(x2) = 0, D(x3) = 0, D(x4) = 0,1, 2, 3,4,D(x5) = 0, 1, 2,3, 4; et quatre contraintes atleast(X, 1,1), atleast(X, 1, 2),Atleast(X, 1,3), et atleast(X, 1,4) ce qui signifie que chaque valeur de {1, 2,3, 4}doit être prise au moins une fois par une variable de X dans toute solution. Il est clair que le problème considère n'a pas de solutions, car quatre valeurs doivent être prise au moins une fois et il n'existe que deux variables pouvant les prendre. Or, si l'on considéré les filtrages associés aux contraintes atmost et atleastPrises individuellement, comme nous avons présente précédemment, on s'aperçoit Qu'aucune valeur n'est supprimée d'un domaine. En effet, les domaines des variablesx4 et x5 restent les mêmes parce que toute valeur de {1, 2,3, 4} appartient `a au moins deux domaines. Pour cet exemple, nous pouvons déduire une nouvelle contrainte à partir de la présence simultanée de plusieurs contraintes. Si quatre valeurs doivent être prises au moins une fois par cinq variables alors les autres valeurs ne peuvent être prises qu'au plus $5 - 4 = 1$ fois, donc nous pouvons ajouter la contrainte atmost(x, 1, 0).En introduisant cette nouvelle contrainte un échec est immédiatement déduit. Cette idée peut ˆêtre généralisée pour n'importe quelle contrainte globale de cardinalité. Soit card (ai) la variable associée à chaque valeur ai de D(X) qui compte le nombre de domaines de X qui contiennent ai. Nous avons li ≤ card (ai) ≤ ui, où liEt ui sont respectivement le minimum et le maximum de fois où la valeur ai doit être Prise, alors, nous pouvons simplement déduire la contrainte Pai∈D(X) card(ai) =|X| ; et chaque fois que le minimum ou le maximum de card (ai) sont modifiés, les

Valeurs de li et lui sont mises à jour et la GCC est modifiée. Cette méthode montre comment on peut définir simplement un filtrage plus puissant en introduisant des contraintes supplémentaire. Bien entendu, la consistance d'arc de la contrainte globale n'est pas assurée, mais la m´méthode est simple et facile à mettre en œuvre. Pour résumer, nous pouvons donc énumérer trois intérêts majeurs pour les contraintes globales :

- L'expressivité : il est plus pratique de définir une contrainte correspondant à un ensemble de contraintes plutôt que de définir indépendamment chacune des contraintes de cet ensemble.

- Comme une contrainte globale correspond à un ensemble de contraintes il est possible de déduire plus d'informations à partir de la présence simultanée de contraintes.

- Des algorithmes de filtrage puissants prenant en compte l'ensemble des contraintes comme un tout peut être écrit. Ces algorithmes de filtrage rendent possible l'utilisation de techniques de recherche opérationnelle ou de théorie des graphes en PPC.

De nombreuses contraintes globales ont ainsi été développées, nous pouvons citer par exemple la contrainte cumulative [20] la contrainte d'ordonnancement d'activités non interruptibles [21], la contrainte diffn [22], la contrainte cycle, la ontrainte sort [23] [24],La contrainte alldiff, la contrainte symétrique alldiff, la contrainte globale de cardinalité, la contrainte globale de cardinalité avec coûts, la contrainte de produit scalaire de variables toutes différentes], la contrainte séquence la contrainte stretch [25], la contrainte globale de distance minimum[26], la contrainte k-diff et la contrainte du nombre de valeurs distinctes [23] enfin, mentionnons l'état de l'art sur l'usage de contraintes globales de H. Simonis [24].

2.4.2. Un exemple de modélisation efficace :

Nous proposons dans cette section de donner brièvement un modèle efficace pour résoudre un problème difficile de calcul de calendrier de tournois sportifs décrit dans, le problème consiste à déterminer les matchs entre n équipes pour n−1 semaines, en sachant que chaque semaine est divisée en n/2 périodes, le but est donc de définir pour chaque période et pour chaque semaine le match qui doit être jouée en tenant compte des contraintes suivantes :

1. Chaque équipé doit jouer contre toutes les autres équipes.
2. Une équipé joue exactement une fois chaque semaine
3. Une équipe ne peut jouer qu'au plus deux fois pendant la saison dans la même Période.

	Sem. 1	Sem. 2	Sem. 3	Sem. 4	Sem. 5	Sem. 6	Sem. 7
Period 1	1 vs 2	1 vs 3	4 vs 8	4 vs 7	4 vs 8	2 vs 6	3 vs 5
Period 2	3 vs 4	2 vs 8	1 vs 4	6 vs 8	2 vs 5	1 vs 7	6 vs 7
Period 3	5 vs 6	4 vs 6	2 vs 7	1 vs 5	3 vs 7	3 vs 8	1 vs 8
Period 4	7 vs 8	5 vs 7	3 vs 6	2 vs 3	1 vs 6	4 vs 5	2 vs 4

Table 2. 1: un exemple efficace de Planifications

La table suivante donne une solution du problème pour 8 équipes :

Ce problème est particulièrement intéressant pour la PPC, tout d'abord parce que c'est un benchmark standard (proposé par Bob Daniel) des problèmes MIP et il est affirmé (cf. [20]) que les meilleurs solvers MIP ne peuvent pas trouver une solution pour 14 équipes, alors que le modèle proposé dans cette section est nettement plus efficace. ensuite, cet exemple met en évidence les caractéristiques fondamentales de la PPC, autrement dit l'utilisation de contraintes globales, en particulier cet exemple utilise la consistance d'arc des contraintes globales de cardinalité .L'idée principale est d'utiliser deux classes de variables : pour une semaine et une période données, les variables d'équipes spécifient l'équipe qui joue et les variables de matchs expriment le match qui est jouée, l'utilisation de variables de matchs rend plus simple l'expression de la contrainte imposant que toutes les équipes doivent se rencontrer une et une seule fois, les matchs sont identifiés sans ambigüité à l'aide des deux équipes qui le compose, plus précisément, un match formé par l'équipe h jouant à domicile et l'équipe a jouant à l'extérieur est identifié par l'entier $h*n+a$, on peut poser une contrainte entre les deux équipes jouant ensemble afin de casser une symétrie. En effet, pour chaque période et pour chaque semaine donnée le problème ne différencie pas le match (a vs b) du match (b vs a), on peut donc imposer la contrainte établissant que seul le match (a vs b) avec $a < b$ doit être envisage.

Les variables d'équipes et les variables de matchs doivent être liés ensemble afin de s'assurer que pour une période et une semaine donnée les valeurs possibles pour La variable de match et celles des deux variables d'équipes sont cohérentes entre elles, ce lien est mis en œuvre à l'aide d'une contrainte dont l'ensemble des tuples est donnée en extension, pour 8 équipes, cet ensemble est constitué des tuples de la forme (1,2, 1) (ce qui signifie que le match opposant les équipes 1 et 2 est le match numéro 1), (1, 3, 2), ..., (7, 8,56), donc pour chaque intersection entre une ligne et une colonne, on définit une telle contrainte Le problème peut être rendu plus uniforme en introduisant une colonne supplémentaire que l'on nomme "extra" colonne. On peut alors modifier la contrainte sur

les lignes imposant qu'une équipe sera présente au plus deux fois par période. En effet, on peut remarquer qu'une équipe doit jouer n − 1 matchs, puisqu'elle a n – 1Adversaires possibles. Or, une équipe ne peut jouer qu'au plus deux fois par période et il y a n/2 périodes, donc une équipe jouera deux fois par période pour toutes les périodes, sauf une pour laquelle elle ne joue qu'une seule fois. Aussi, pour chaque Période il y a toujours exactement deux équipes qui ne jouent qu'une et une seule fois pour cette période. Donc, si l'on place pour chaque période ces deux équipes dans la colonne "extra", on peut changer la contrainte sur les périodes : en incorporant la colonne "extra" chaque équipe doit jouer exactement deux fois par période. En Outre, chaque équipe n'est introduite dans la colonne "extra" qu'une et une seule fois, donc pour cette colonne on introduit une nouvelle contrainte alldiff. Ainsi, Nous avons pour chaque période une contrainte globale de cardinalité impliquant Toutes les variables d'équipes de la période (la colonne "extra" étant incluse) et imposant que toute équipe doit être prise exactement deux fois par ces variables. Pour chaque semaine, nous définissons une contrainte alldiff impliquant toutes les variables d'équipes de cette semaine. On introduit également une contrainte alldiff pour la colonne "extra». La stratégie de sélection d'affectation consiste à sélectionner l'équipe la plus Affectée puis à l'affecter à la variable ayant le moins de valeur possible et contenant l'équipe choisie dans son domaine. On obtient alors les résultats suivants (les temps sont

#équipes	8	10	12	14	16	18	20	22	24
#échecs	32	417	41	3,514	1,112	8,756	72,095	6,172,672	6,391,470
temps	0.1	0.3	0.1	0.1	1.5	12	110	3h	4h

Table 2. 2: Les resultas

exprimés en secondes sauf mention particulière) :

Pour ce modèle il est indispensable d'utiliser des algorithmes de filtrage puissants, notamment ceux réalisant la consistance d'arc pour les contraintes globales. Un modèle encore plus performant (le problème avec 40 équipes est résolu) est proposé dans. Cet autre modèle propose d'engendrer D'abord un calendrier puis d'essayer de satisfaire les contraintes de périodes. On s'affranchit ainsi des contraintes de calendrier, autrement dit des contraintes sur Les colonnes et de la contrainte imposant que chaque équipe doit rencontrer chaque autre équipe exactement une fois.

3. Les solveurs existants

3.1. Logiciels académiques

3.1.1. Prolog III (1989)

Le langage de programmation Prolog III est une extension de Prolog au niveau de ce qu'il a de plus fondamental, le mécanisme d'unification, il intègre dans ce mécanisme un traitement fin des arbres et des listes, un traitement numérique et un traitement du calcul propositionnel complet. L'auteur présente ici les spécifications et le modèle logico-mathématique de ce nouveau langage. A cette occasion, il remplace la notion même d'unification par celle plus appropriée de résolution de contraintes, puis il illustre les possibilités accrues du langage à travers des exemples variés.

3.2. Logiciels commerciaux

3.2.1. IBM CP Optimize

CPLEX est un outil informatique d'optimisation commercialisé par IBM depuis son acquisition de l'entreprise française ILOG en 2009. Son nom fait référence au langage C et à l'algorithme du simplexe. Il est composé d'un exécutable (CPLEX interactif) et d'une bibliothèque de fonctions pouvant s'interfacer avec différents langages de programmation : C, C++, C#, Java et Python.

3.2.2. Artelys Knitro

Est un solveur d'optimisation commercial spécialisé dans la résolution de problèmes d'optimisation non linéaire.KNITRO – (nom initial) pour "Nonlinear Interior point Trust Region Optimization" (le "K" est silencieux) – a été co-fondé par Richard Waltz, Jorge Nocedal, Todd Plantenga et Richard Byrd. La première version sortie en 2001 est née des travaux de recherche menés au sein de l'université de Northwestern (par le biais de Ziena Optimization LLC) et est désormais développé par Artelys.Les problèmes d'optimisation doivent être passés au solveur sous leur forme mathématique et il est préférable de lui fournir un moyen de calculer les dérivées sous forme de matrice creuse. Knitro peut approximer les dérivées du problème mais les performances sont généralement accrues lorsque les dérivées exactes sont fournies. Il est également possible d'utiliser un langage de modélisation qui calculera automatiquement les dérivées et appellera Knitro depuis l'environnement de modélisation.

3.2.3. CHIP (Constraint Handling in Prolog)

Est un langage de programmation à logique de contrainte développé par M. Dincbas et alias en 1985 à ECRC, utilisant initialement une interface de langage Prolog. CHIP V5 est la version développée et commercialisée par COSYTEC à Paris depuis 1993 avec Prolog,

utilisant des interfaces de langage C, C ++ ou Prolog. Le succès commercial d'ILOG Solver découle également en partie de la version ECRC de CHIP.

3.2.4. Comet

Est un langage de programmation commercial conçu par le professeur de l'Université Brown, Pascal Van Hentenryck, utilisé pour résoudre des problèmes d'optimisation combinatoire complexes dans des domaines tels que l'affectation des ressources et la planification. Il propose toute une gamme d'algorithmes d'optimisation : de la programmation mathématique à la programmation par contraintes, en passant par l'algorithme de recherche local et l'optimisation combinatoire stochastique dynamique, les programmes de comètes spécifient deux algorithmes de recherche locaux : un modèle de haut niveau décrivant les applications en termes de contraintes, de combinateurs de contraintes et de fonctions objectives ; Une procédure de recherche exprimée en termes de modèle à un niveau d'abstraction élevé. Cette approche favorise la réutilisation des applications. Son API lui permet d'être utilisé comme une bibliothèque de logiciels, comet propose également des abstractions de haut niveau pour l'informatique parallèle et distribuée, basées sur la planification des boucles, les interruptions et le vol de travail.

3.2.5. Disolver (Bibliothèque C++)

Est un moteur d'optimisation basé sur des contraintes C ++. Il peut être combiné avec n'importe quelle bibliothèque MPI, pour fonctionner de manière transparente sur des architectures parallèles. Il effectue : Recherche complète. Les méthodes de cette catégorie effectuent une exploration par arborescence d'un espace de recherche problématique. Disolver fournit des algorithmes pour le test de satisfaction et l'optimisation. Recherche parallèle Cette catégorie généralise les algorithmes précédents, par exemple la branche parallèle et la liaison. Il fournit des contrôles avancés d'équilibrage de charge et de partage des connaissances.

3.3. Logiciels sous licences libres :

3.3.1. AbsCon(Bibliothèque Java)

Est le solveur qui a été soumis au Concours 2006 des solveurs CSP. Ce solveur reconnaît le format XML, XCSP 2.0, utilisé pour représenter les instances CSP

3.3.2. Choco

Choco est une bibliothèque Java libre et open-source dédiée à la programmation par contraintes. Il vise à décrire des problèmes combinatoires durs sous la forme de problèmes de satisfaction de contraintes et à les résoudre avec la programmation par contraintes techniques.

3.3.3. Cream

Est une bibliothèque de classes qui aide les programmeurs Java à développer des programmes intelligents nécessitant une satisfaction ou une optimisation des contraintes sur des applications.

3.3.4. Gercode(pour l'environnement de développement de contraintes génériques)

Est une bibliothèque de logiciels pour résoudre les problèmes de satisfaction de contraintes. Il est programmé en C ++ et distribué en tant que logiciel libre sous la licence MIT permissive. Gercode possède des liaisons pour plusieurs langages de programmation tels que Prolog, Python et Ruby, ainsi qu'une interface vers le langage de modélisation AMPL.

3.3.5. Google OR-Tools

Fournit des encapsuleurs de langage de programmation pour les outils de recherche opérationnelle tels que l'optimisation et la résolution de contraintes.

3.3.6. JaCoP

Est un solveur de contraintes pour les problèmes de satisfaction de contraintes. Il est écrit en Java et est fourni en tant que bibliothèque Java. JaCoP a une interface avec les langages de modélisation MiniZinc et AMPL. Son objectif principal est la facilité d'utilisation, la puissance de modélisation et l'efficacité. Il dispose d'une large collection de contraintes globales implémentées pour faciliter la modélisation des problèmes. JaCoP est activement développé depuis 2001.

Autre Solveur : Minion C++, Scarable (Scala), Jopt.

4. Conclusions :

Dans ce Chapitre Nous avons Présenté le paradigme de la Programmations par Contraintes en générale, historique de La PPC et les domaines de L'applications et Nous avons présenté Les Principes De Base de La PPC et Mentionné tous Les Solveur Existant actuellement.

CHAPITRE 3 LA PPC POUR RESOUDRE LE PROBLEME DE PLANIFICATION D'UN LIGUE

1. Introduction

Herrera-Diaz (1997) Décrit la programmation par contraintes (CP) comme une technologie logicielle croissante utilisée pour résoudre des problèmes d'optimisation complexes, généralement combinatoires, contrairement aux autres langages de programmation où le programmeur code une série exacte d'instructions à exécuter, dans CP les contraintes il suffit de définir les caractéristiques de la solution, des problèmes de satisfaction de contraintes sont constitués de variables et de contraintes par lesquelles les contraintes Définir les valeurs que les variables peuvent prendre.

Dans ce chapitre nous allons formaliser le problème de planification d'une ligue sportive qui est un problème d'optimisation comme étant un problème de satisfaction de contrainte que l'on peut résoudre en utilisant la PPC Nous présentons aussi le solveur que nous allons utiliser pour résoudre notre problème et les Contrainte programmé.et dans 2^{éme} partie de ce chapitre nous présentons l'implémentation de notre applications et nous présentons ces différentes fonctionnalités.

2. Notre démarche pour la résolution du problème de planification d'une ligue

Dans cette section, nous présentons notre démarche déclarative pour la résolution du problème de planification d'une ligue sportive basée sur la programmation par contraintes. Une démarche déclarative en ce sens que nous spécifions quelle planification doit être faite, au lieu de spécifier de manière algorithmique comment elle doit être trouvée. Dans notre démarche, le problème exprimé en termes de contraintes et un solveur générique trouve les solutions, notre méthode basé sur trois étapes : décrire, modéliser, exécuter pour trouver une solution à un problème de mise au point d'un planning pour une ligue sportive..

2.1. Présentations de Notre Problème :

La première étape de la résolution du problème de mise au point d'un planning pour une ligue sportive consiste à décrire ce problème en langage naturel.

Dans cette leçon, vous allez résoudre un problème de mise au point d'un planning pour une ligue sportive. La ligue compte 10 équipes qui jouent des matchs au cours d'une saison de 18 semaines. Chaque équipe dispose d'un terrain à domicile et rencontre les autres équipes de la ligue à deux reprises au cours de la saison, une fois à domicile et une fois chez l'équipe adverse. Pour chaque match, l'équipe qui joue sur son propre terrain est appelée "équipe à domicile", tandis que l'équipe qui joue chez l'équipe adverse est appelée "équipe en déplacement". 90 matchs au total sont prévus. Une interruption correspond à une séquence de semaines consécutives pendant lesquelles une équipe joue tous ses matchs soit à domicile, soit en déplacement. Aucune équipe ne peut avoir d'interruption de plus de deux semaines. L'objectif de ce problème est de minimiser le nombre total d'interruptions des équipes

2.2. Modélisations de notre problème

a première étape de la modélisation et de la résolution d'un problème consiste à décrire le problème en langage naturel, en identifiant les variables de décision et les contraintes placées sur ces variables.

Décrivez le problème en langage naturel. Répondez à ces questions :

- Quelles sont les informations connues dans ce problème ?
- Quelles sont les variables ou les inconnues de ce problème ?
- Quelles sont les contraintes placées sur ces variables ?
- Quel est l'objectif ?

Quelles sont les informations connues dans ce problème ?

- 10 équipes sont présentes.

- Ces équipes se rencontrent sur une saison de 18 semaines. Chaque semaine compte 5 créneaux pour les matchs.
- Chaque équipe doit jouer contre les autres équipes à deux reprises, dont une fois au domicile de chaque équipe. Les 90 matchs portent chacun un identificateur unique compris dans la plage 0..89.

Quelles sont les contraintes placées sur ces variables ?

- Chaque équipe doit jouer une seule fois par semaine.
- Chaque match ne doit être joué qu'une seule fois.
- Pour chaque paire d'équipes, les deux équipes s'opposent deux fois dans une saison et ces deux matchs doivent être planifiés dans différentes parties de la saison. En outre, les deux matchs doivent être planifiés à six semaines d'intervalle au minimum.
- Aucune équipe ne peut avoir d'interruption de plus de deux semaines.
- Chaque équipe doit jouer un seul de ses premiers et derniers matchs à domicile.

Quelles sont les fonctions utilisées pour modéliser les contraintes ?

- utiliser des contraintes définies à partir de tuples autorisés ;
- utiliser les contraintes spécialisées IloInverse et IloAllDiff ;
- utiliser les fonctions IloDiv et IloAbs ;
- utiliser les contraintes logiques ;
- utiliser des contraintes supplémentaires pour améliorer les performances de résolution ;
- utiliser des phases de recherche pour affiner la stratégie de recherche.

Vous allez apprendre à modéliser et à résoudre un problème d'affectation relatif à la planification de matchs pour une ligue sportive. Pour trouver la solution optimale à ce problème avec CP Optimizer, vous utilisez une méthode en trois étapes : décrire, modéliser et exécuter

2.3. Les Solveur ILOG (Notre Choix) :

Le Solveur ILOG sera l'outil utilisé pour trouver une solution au problème car il est fourni une documentations et support pour simplifier utilisations, Il est composé d'un exécutable (CPLEX interactif) et d'une bibliothèque de fonctions pouvant s'interfacer avec différents langages de programmation : C, C++, C#, Java et Python.

Figure 3. 1: logo de ILOG CPLEX

2.3.1. Présentation et Caractéristiques :

CPLEX est, à la base, un solveur de programmes linéaires. Il est commercialisé par la Société ILOG depuis la version 6.0. La dernière version, à ce jour, est la version 11.0. Les Composants de la suite d'optimisation ILOG sont illustrés dans la Figure 3.2.

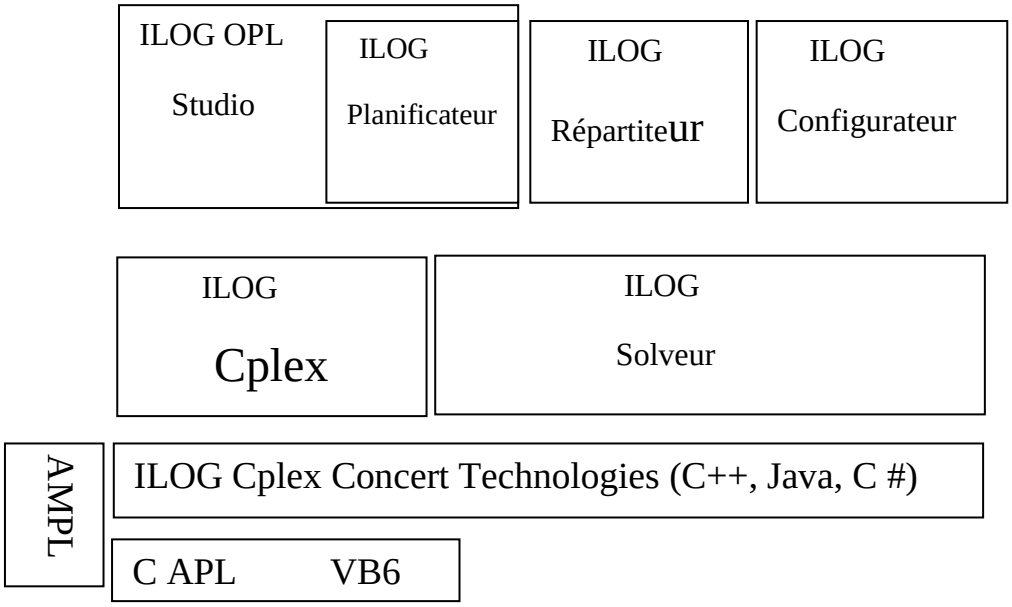

Figure 3. 2: Suite d'optimisation

- *ILOG CPLEX :*Le cœur du système résout des problèmes de programmation mathématique.
- *ILOG Solveur :*La partie principale du système résout des applications en utilisant la programmation par contraintes.
- *ILOG concert technologies :*Les bibliothèques contiennent les fonctionnalités du système, Elles sont disponibles pour les langages C++, Java et .NET.
- *ILOG Planificateur :*Fournit des extensions pour résoudre des problèmes de planification.
- *ILOG Répartiteur :* Fournit des extensions pour la résolution de problèmes de tournées de véhicules.
- *ILOG Configurateur :* Ce module contient des utilitaires pour l'optimisation des ventes en ligne (problèmes de e-commerce).
- *ILOG OPL Studio :*OPL est un langage pour la modélisation des problèmes d'optimisation, Il interagit directement avec les modules ILOG cplex, ILOG Solveur et ILOG Répartiteur**.**
- *AMPL :*C'est un autre langage pour la modélisation, qui interagit avec le module ILOG CPLEX (AMPL a été développé par les laboratoires Bell).
- *C et VB6 APIs :*Ce sont des bibliothèques pour des utilisateurs du langage C et de l'environnement VB6. Elles permettent de les interfacer avec le module ILOG CPLEX.

2.3.2. Historique :

CPLEX a été initialement développé par l'équipe de Robert Bixby pour disposer d'un Solveur performant pour résoudre des instances du problème de voyageur de commerce de grande taille. Jusqu'à la version 6.0, il a été Commercialisé par la société CPLEX. En 1996, cette société a été rachetée par ILOG pour étoffer son éventail de produits destinés à l'Aide à la Décision.

2.3.3. Capacités :

Initialement, CPLEX est un solveur de programmes linéaires. A ce titre, il repose donc Sur une implémentation performante du simplexe primal. Il dispose également du simplexe Dual et du simplexe de réseau. Il peut aussi résoudre des programmes linéaires mixtes, en Combinant le simplexe, le branch and bound et la génération de coupes. Depuis peu, il intègre Également une technique à base de points intérieurs et peu traité des problèmes quadratiques. Actuellement, CPLEX est un des solveurs les plus performants disponibles,

sinon le plus Performant. Il peut ainsi traiter des problèmes contenant plusieurs dizaines de milliers de Variables et plusieurs centaines de milliers de contraintes. Pour les problèmes mixtes, la limite Est sensiblement plus basse, mais elle dépend grandement du type de problèmes et du modèle appliqué. Les problèmes traités par la suite d'optimisation ILOG sont : les programmes linéaires et linéaires mixtes, les programmes quadratiques et quadratiques mixtes, les programmes avec Contraintes quadratiques et avec contraintes quadratiques mixtes.

2.3.4. Modes d'utilisation :

Il existe deux manières d'utiliser CPLEX. La première consiste à travailler de manière interactive en invoquant un interpréteur de commande dédié. La seconde consiste à appeler directement les fonctionnalités du moteur depuis son propre code, que ce soit du C, du C++ou du Java.

3. Présentation de L'environnement de développement

On utilise Netbeans IDE pour un environnement de Travail avec Bibliothèque ILOG comme un solveur pour trouver la solution de Notre Problème.

Figure 3. 3: Logo de netbeans

3.1. Historique :

NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL (Common Développent and Distribution License) et GPLv2. En plus de Java, NetBeans permet la prise en charge native de divers langages tels le C, le C++, le JavaScript, le XML, le Groovy, le PHP et le HTML, ou d'autres (dont Python et Ruby) par l'ajout de greffons. Il offre toutes les facilités d'un IDE moderne

(éditeur en couleurs, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

Compilé en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). Un environnement Java Development Kit JDK est requis pour les développements en Java. NetBeans constitue par ailleurs une plateforme qui permet le développement d'applications spécifiques (bibliothèque Swing (Java)). L'IDE NetBeans s'appuie sur cette plateforme. L'IDE Netbeans s'enrichit à l'aide de greffons.

3.2. Environnement :

Netbeans est un IDE qui supporte une large variété de langages de programmation et d'outils de collaboration.

3.3. Environnement de base :

L'environnement de base comprend les fonctions générales suivantes :

- configuration et gestion de l'interface graphique des utilisateurs,
- support de différents langages de programmation,
- traitement du code source (édition, navigation, formatage, inspection),
- fonctions d'import/export depuis et vers d'autres IDE, tels qu'Eclipse ou JBuilder,
- accès et gestion de bases de données, serveurs Web, ressources partagées,
- gestion de tâches (à faire, suivi…),
- documentation intégrée.

3.4. Principaux langages supportés :

L'éditeur intégré propose des fonctions de complètement, de contrôles syntaxiques et sémantiques, d'avertissements et de conseils, de reprises de code (« refactoring » : renommage, changement des méthodes, gestion des classes, …), de sauvegarde et reprise. Il supporte principalement les langages suivants5 :

- Java (Java SE6, Java ME7, Java FX8, Java EE9), Javadoc ;
- Groovy et Grails10 ;
- PHP (dont les environnements Zend et Symfony)11 ;
- JavaScript12 ;

- C, C++, Fortran13. Netbeans ne requiert pas l'utilisation d'un compilateur particulier. À noter le support des bibliothèques Qt. Les plates formes supportées sont Microsoft Windows, Linux, Mac OS, Solaris 10 et OpenSolaris ;
- Python14 (via un greffon développé par la communauté) ;
- HTML, XHTML, RHTML (en) ;
- XML ;
- DTD ;
- CSS ;
- JSP, JSF;

4. L'implémentations :

4.1. Langage De Programmations :

*Java :*Java est un langage interprété, ce qui signifie qu'un programme compilé n'est pas directement exécutable par le système d'exploitation mais il doit être interprété par un autre programme, qu'on appelle interpréteur.

Un programmeur Java écrit son code source, sous la forme de classes, dans des fichiers dont l'extension est .java . Ce code source est alors compilé par le compilateur javac en un langage appelé bytecode et enregistre le résultat dans un fichier dont l'extension est .class. Le byecode ainsi obtenu n'est pas directement utilisable. Il doit être interprété par la machine virtuelle de Java qui transforme alors le code compilé en code machine compréhensible par le système d'exploitation. C'est la raison pour laquelle Java est un langage portable : le bytecode reste le même quel que soit l'environnement d'exécution.

Figure 3. 4: Logo de java

4.2. Les principaux codes source :

Notre Applications Contient une seul Classe Principale et Trois Forme :

- Classe Sport
- Forme Home
- Forme Plan_Equipes
- Forme Plan_Matchés

Les Contraintes Programmé :

Pour chaque slot de jeu, configurez la correspondance entre l'identifiant du jeu, Équipe domicile et équipe à l'extérieur

```
IloIntTupleSet gha = cp.intTable(3);
int[] tuple = new int[3];
for (int i = 0; i < n; i++) {
    tuple[0] = i;
    for (int j = 0; j < n; j++) {
        if (i != j) {
            tuple[1] = j;
            tuple[2] = Game(i, j, n);
            cp.addTuple(gha, tuple);
        }
    }
}
```

Figure 3. 5: Code du l'identifiant

Toutes les équipes jouent chaque semaine

```
for (int i = 0; i < nbWeeks; i++) {
    IloIntVar[] teamsThisWeek = cp.intVarArray(n);
    for (int j=0; j < nbGamesPerWeek; j++) {
        teamsThisWeek[j]= home[i][j];
        teamsThisWeek[nbGamesPerWeek+j]=away[i][j];
    }
    cp.add(cp.allDiff(teamsThisWeek));
}
```

Figure 3. 6: Code du contraint joue chaque semaine

Double représentation : pour chaque identifiant de jeu, la fente de lecture est maintenue

```
IloIntVar[] weekOfGame= cp.intVarArray(nbGames, 0, nbWeeks - 1);
IloIntVar[] allGames= cp.intVarArray(nbGames);
IloIntVar[] allSlots= cp.intVarArray(nbGames, 0, nbGames - 1);
for (int i = 0; i < nbWeeks; i++)
    for (int j = 0; j < nbGamesPerWeek; j++)
        allGames[i*nbGamesPerWeek+j]=games[i][j];
cp.add(cp.inverse(allGames, allSlots));
for (int i = 0; i < nbGames; i++)
    cp.add(cp.eq(weekOfGame[i], cp.div(allSlots[i], nbGamesPerWeek)));
//
```

Figure 3. 7: Code du contraint fente

Deux demi-horaires. Ne peut pas jouer la même paire deux fois dans la même moitié. De plus, imposer un nombre minimum de semaines entre deux jeux impliquant les mêmes équipes (jusqu'à six semaines)

```
int mid = nbWeeks / 2;
int overlap = 0;
if (n >= 6)
    overlap = min(n / 2, 6);
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        int g1 = Game(i, j, n);
        int g2 = Game(j, i, n);
        cp.add(cp.equiv(cp.ge(weekOfGame[g1], mid) , cp.lt(weekOfGame[g2], mid)));
        // Six week difference...
        if (overlap != 0)
            cp.add(cp.ge(cp.abs( cp.diff(weekOfGame[g1], weekOfGame[g2])), overlap));
    }
}
```

Figure 3. 8: Code du contraint d'horaire

Ne peut pas avoir trois jeux à domicile ou trois déplacements.

```
IloIntVar[][] playHome = new IloIntVar[n][];
for (int i = 0; i < n; i++) {
    playHome[i] = cp.intVarArray(nbWeeks, 0, 1);
    for (int j = 0; j < nbWeeks; j++)
        cp.add(cp.eq(playHome[i][j], cp.count(home[j], i)));
    for (int j = 0; j < nbWeeks -3; j++) {
        IloIntVar[] window = cp.intVarArray(3);
        for (int k = j; k < j + 3; k++)
            window[k-j]=playHome[i][k];
        IloIntExpr windowSum = cp.sum(window);
        cp.add(cp.ge(windowSum, 1));
        cp.add(cp.le(windowSum, 2));
    }
}
```

Figure 3. 9: Code du contraint maison

Si nous commençons la saison à la maison, nous finissons à aller et inversement.

```
for (int i = 0; i < n; i++)
    cp.add(cp.neq(playHome[i][0], playHome[i][nbWeeks-1]));
```

Figure 3. 10: Code du contrainte commençons la saison à la maison

Objectif: minimiser le nombre de "ruptures". Une pause est deux matchs consécutifs à domicile ou à l'extérieur pour une équipe particulière

```
IloIntVar[] teamBreaks= cp.intVarArray(n, 0, nbWeeks / 2);
for (int i = 0; i < n; i++) {
    IloIntExpr nbreaks= cp.constant(0);
    for (int j = 1; j < nbWeeks; j++)
        nbreaks = cp.sum(nbreaks,
                         cp.intExpr(cp.eq(playHome[i][j-1],
                                          playHome[i][j])));
    cp.add(cp.eq(teamBreaks[i], nbreaks));
}
IloIntVar breaks = cp.intVar(n - 2, n * (nbWeeks / 2));
cp.add(cp.eq(breaks, cp.sum(teamBreaks)));
cp.add(cp.minimize(breaks));
```

Figure 3. 11: Code du contrainte de minimisation des ruptures

Chaque équipe joue à domicile le même nombre de fois

```
for (int i = 0; i < n; i++)
    cp.add(cp.eq(cp.sum(playHome[i]),nbWeeks / 2));

// Breaks must be even for each team
for (int i = 0; i < n; i++)
    cp.add(cp.eq(cp.modulo(teamBreaks[i], 2),0));
```

Figure 3.13 : Code du contrainte de minimisation des ruptures

Les équipes sont interchangeables. Fix première semaine. Brise également la symétrie de réflexion de tout le programme.

```
for (int i = 0; i < nbGamesPerWeek; i++) {
    cp.add(cp.eq(home[0][i], i * 2));
    cp.add(cp.eq(away[0][i], i * 2 + 1));
}
```

Figure 3.14 : Code du contrainte équipe interchangeables

L'ordre des jeux de chaque semaine est arbitraire. Briser la symétrie en forçant une commande.

```
for (int i = 0; i < nbWeeks; i++)
    for (int j = 1; j < nbGamesPerWeek; j++)
        cp.add(cp.gt(games[i][j], games[i][j-1]));

cp.setParameter(IloCP.DoubleParam.TimeLimit, 20);
cp.setParameter(IloCP.IntParam.LogPeriod, 10000);
IloVarSelector varSel = cp.selectSmallest(cp.varIndex(allGames));;
IloValueSelector valSel = cp.selectRandomValue();

IloSearchPhase phase = cp.searchPhase(allGames,
                                      cp.intVarChooser(varSel),
                                      cp.intValueChooser(valSel));
cp.startNewSearch(phase);
```

Figure 3. 12: Code du contrainte ordre du jeux arbitraire

Trouver Solution Des Planifications des Matches :

```
while (cp.next()) {
    System.out.println("Solution at " + (int)cp.getValue(breaks));
     System.setOut(new PrintStream(new FileOutputStream("resultat2.txt")));
    for (int j = 0; j < nbWeeks; j++) {
        System.out.print("Week" + j+ ":");

        if ( j < 10 ) System.out.print("");
        for (int i = 0; i < nbGamesPerWeek; i++) {
            int h = (int) cp.getValue(home[j][i]);
            int a = (int) cp.getValue(away[j][i]);
            if (h >= 10) System.out.print(h);

            else System.out.print("/" + h);
            System.out.print("-");

            if (a >= 10)  System.out.print(a);
            else          System.out.print(a + "");
            System.out.print(" ");

        }
        System.out.println();


    }
```

Figure 3. 13: Code des planifications des matchs

Trouver Solutions Des Planifications Des équipes :

```
        System.out.println( "Team schedules");
        for (int i = 0; i < n; i++) {
            System.out.print("T"+ i + ":/");
            int prev = -1;
            int brks = 0;
            for (int j = 0; j < nbWeeks; j++) {
                for (int k = 0; k < nbGamesPerWeek; k++) {
                    if (cp.getValue(home[j][k]) == i) {
                        int t = (int) cp.getValue(away[j][k]);
                        if (t >= 10)System.out.print(t + "H/");
                        else         System.out.print(" " + t + "H/");
                        if (prev == 0) brks ++ ;
                        prev = 0;
                    }
                    if (cp.getValue(away[j][k]) == i) {
                        int t = (int) cp.getValue(home[j][k]);
                        if (t >= 10) System.out.print(t + "A/");
                        else          System.out.print(" " + t + "A/");

                        if(prev == 1) brks ++;
                        prev = 1;
                    }
                }
            }
            System.out.println("" + brks + " breaks");

        }
    }
    cp.endSearch();
catch (IloException e) {
```

Figure 3. 14: Code du planification des équipes

Code Pour enregistrer une solution Dans Fichier.txt :

```
System.setOut(new PrintStream(new FileOutputStream("resultat2.txt")));
```

Figure 3. 15: Code d'enregistrement

Code Pour Récupérer La solution depuis Le Fichier texte et Afficher Dans Tableau :

```
    filePath ="C:\\Users\\Amine\\Desktop\\resultat2.txt";
File file= new File(filePath);
    try {
        FileReader fr = new FileReader(file);
        BufferedReader br = new BufferedReader(fr);
        DefaultTableModel model = (DefaultTableModel)jTable2.getModel();
        Object [] tableLines =br.lines().toArray();
        for(int i=19;i<tableLines.length;i++){

            String[] dataRow= tableLines[i].toString().split("/");
            model.addRow(dataRow);
        }
    } catch (FileNotFoundException ex) {
        Logger.getLogger(Plan_Equipes.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Figure 3. 16: Code du récupération de la solution depuis le fichier

Code Pour Imprimer Résultat de Table De Planifications :

```
MessageFormat header = new MessageFormat("Welcome to Team Shedules");
  MessageFormat Footer = new MessageFormat("Page{0,number,integer}");
   try {
       jTable2.print(JTable.PrintMode.NORMAL, header, Footer);
   } catch (Exception e) {

   }
```

Figure 3. 17: Code d'impression du table

5. Présentations de l'Applications et Ses fonctionnalités :

5.1. Interface Principale Home :

Contient 2 Partie :

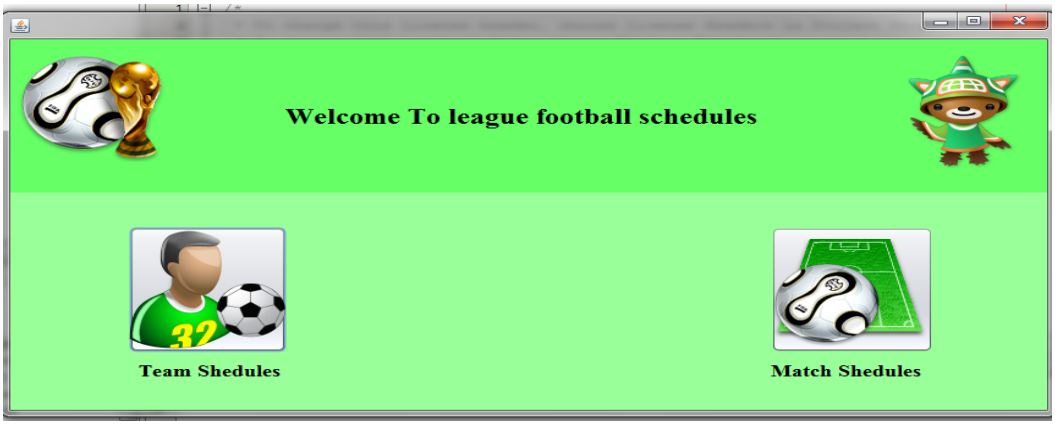

Figure 3. 18: Interface principale

- Partie 1 : pour Afficher La forme Pl_Matches qui Affiche Le Tableau de Planifications des Matches

Figure 3. 19: Planifications des matches

- Partie 2 : Pour Afficher La forme PL_Equipes qui Affiche Le Tableau de Planification Des Equipes

Figure 3. 20: Planifications des équipes

5.2. Interface de Planifications Des Matches (pl_Match.java) :

Contient Une Tableau et 3 Boutons :

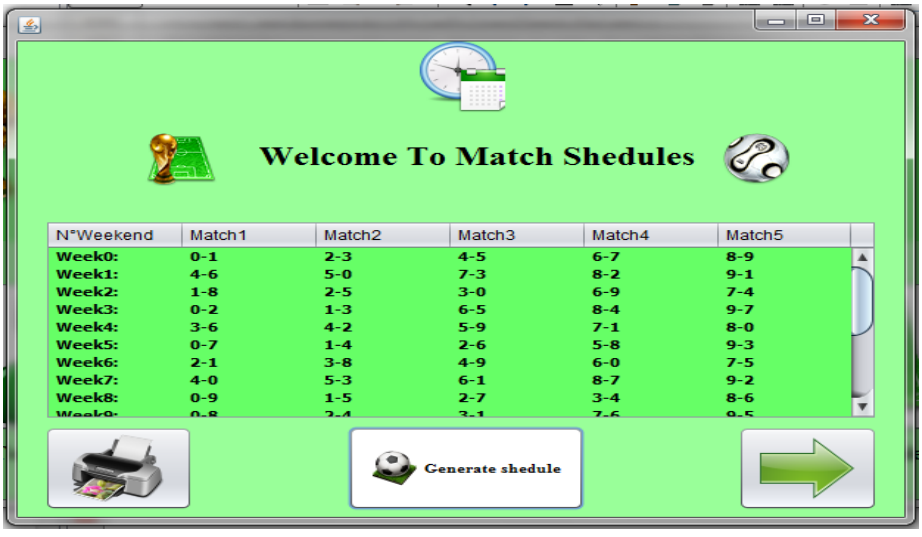

Figure 3.23 : Interface de planifications des matches

Le Tableau Contient résultat De Planifications

- Bouton 1 : Pour retourne a La Forme Principale et fermer La Forme Actuelle

Figure 3. 21: Retourne

- Bouton 2 : Pour Générer La solutions de Planifications des Matche Dans Le tableau

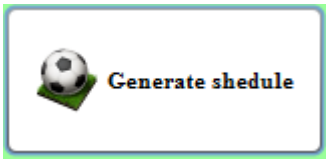

Figure 3. 22: Générer un calendrier

- Bouton 3 : Pour Imprimer résultat de planification ou Enregistrer Dans une fichier PDF

Figure 3. 23: Impression

5.3. Interface de Planifications Des Equipes (pl_Equipes.java) :

Contient une Tableau et trois Boutons .

Figure 3. 24: Interface de planifications des équipes

- Bouton 1 : Pour retourne a La Forme Principale et fermer La Forme Actuelle

Figure 3. 25: Retourne

- Bouton 2 : Pour Générer La solutions de Planifications Des Equipes Dans Le tableau

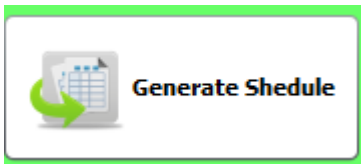

Figure 3. 26: Générer un calendrier

- Bouton 3 : Pour Imprimer résultat de planification ou Enregistrer Dans une fichier PDF

Figure 3. 27: Impression

5.4. Présentations Des Résultats Obtenus

Notre Solveur de IBM trouve 4 Solutions Possibles :

On va Consulter les 4 Solution Dans les Figures Suivantes :

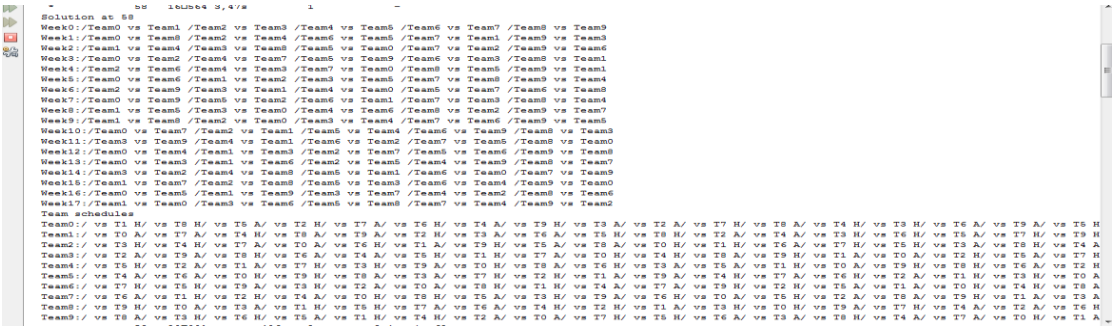

Figure 3. 28: Solution 1

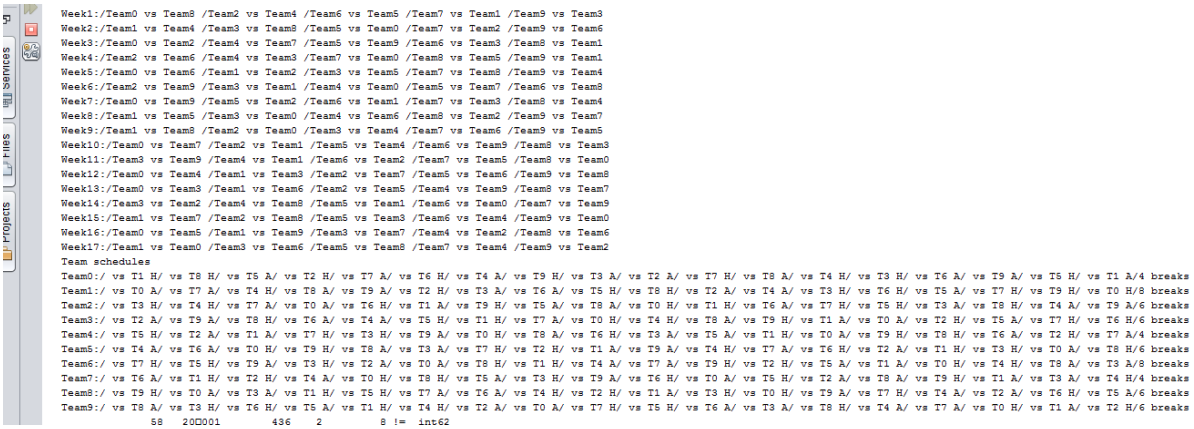

Figure 3. 29: Solution 2

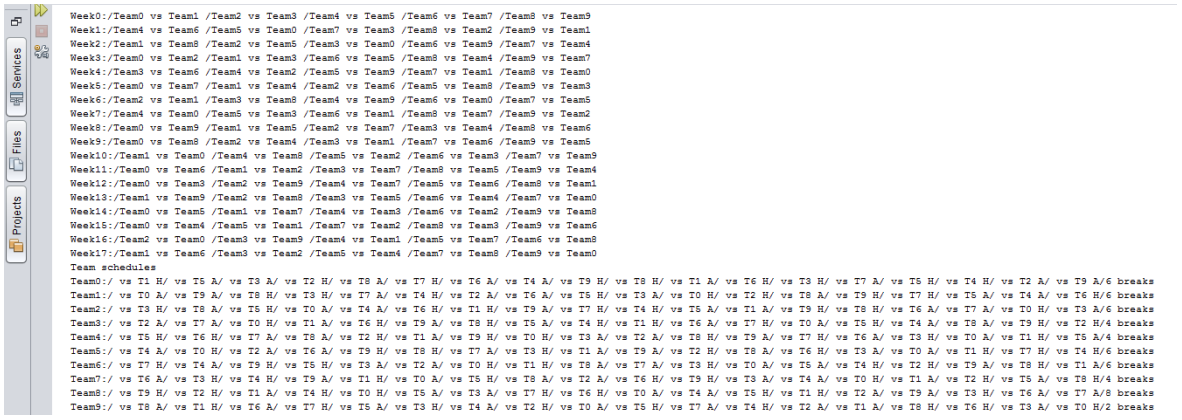

Figure 3. 30: Solution 3

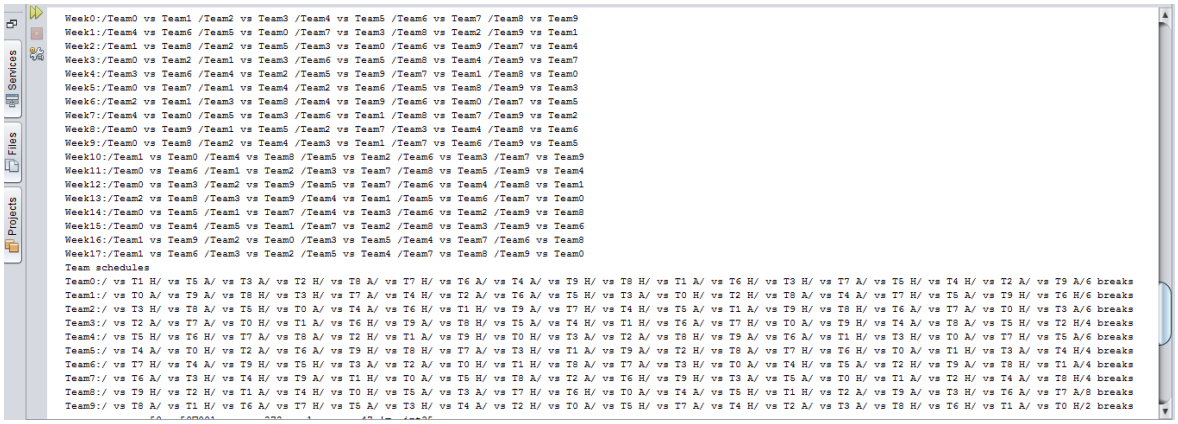

Figure 3. 31: Solution 4

Notre Applications Sauvegarde Une Seule Solutions Dans Une Fichier (Database.txt)

Et récupéré la solution et afficher Dans une Tableau.

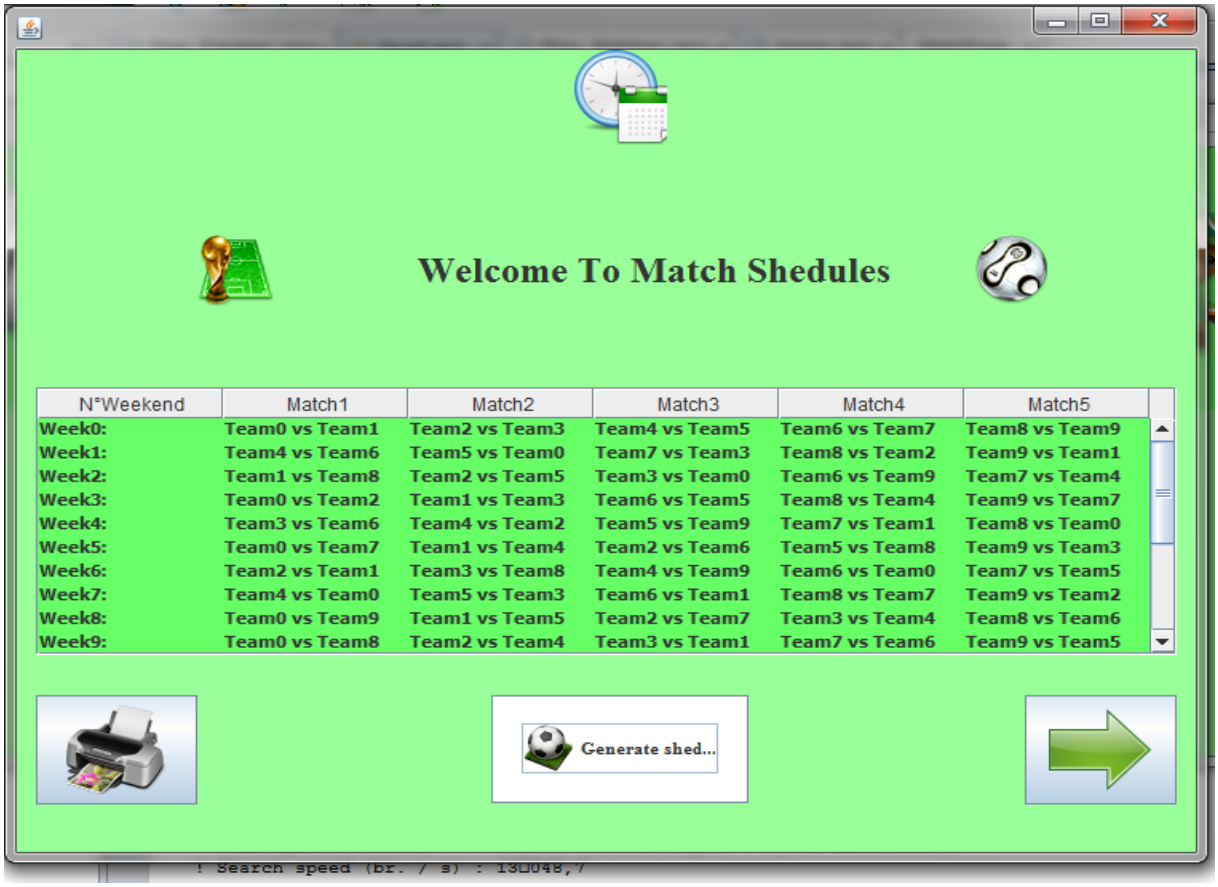

Figure 3. 32: résultat des Planifications des Matches

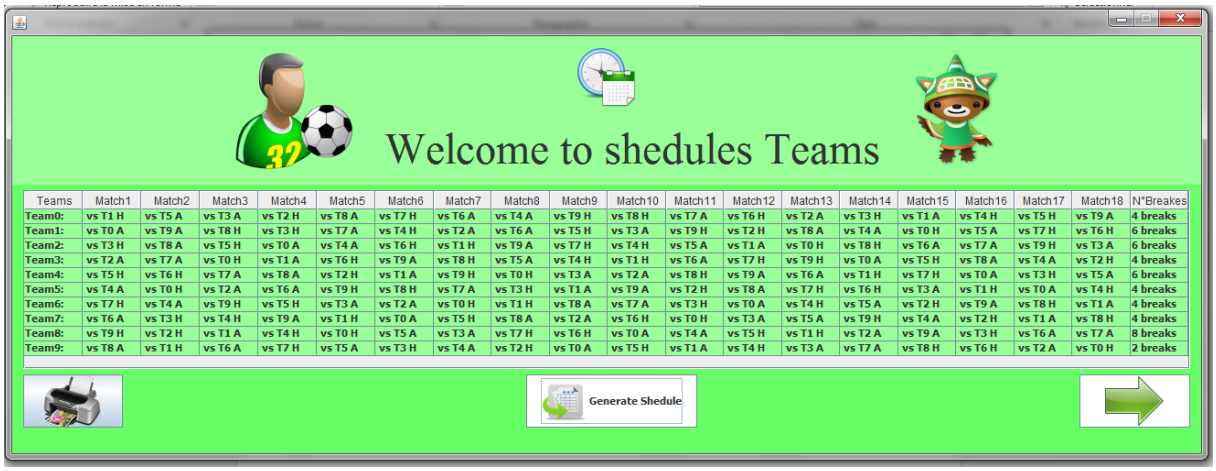

Figure 3. 33: Résultat des Planifications Des Equipes

6. Conclusions :

Dans Première Partie de Ce Chapitre Nous avons Présenté le fonctionnement et Les Caractéristiques de Notre Solveur et Nous avons mentionné autre Solveur Existant, Dans Deuxième Partie de Ce Chapitre nous avons présenté les étapes de la mise en œuvre de notre projet avec tous les outils, les langages et les plateformes utilisés ainsi que la présentation avec l'explication du rôle de chaque outil. Nous avons illustré les interfaces graphiques avec une description textuelle.

CONCLUSION GENERALE

On peut constater que la planification d'horaires est un processus complexe, Cette complexité s'accroît considérablement dans les modélisations des contraintes et prendre en compte les exigences des équipes et les parties prenantes.

Parmi tous les types de plannings, c'est sur les plannings sportifs que nous allons porter notre intérêt, et tout particulièrement sur les plannings des matches entre les équipes et nous avons présenté quelques approches existantes pour résoudre ce problème.

Nous avons Présenté le paradigme de la Programmations par Contraintes en générale, historique de La PPC et les domaines de L'applications et Nous avons présenté Les Principes De Base de La PPC et Mentionné tous Les Solveur Existant actuellement.

Nous avons Présenté le fonctionnement et Les Caractéristiques de Notre Solveur

Et nous avons présenté les étapes de la mise en œuvre de notre projet avec tous les outils, les langages et les plateformes utilisés ainsi que la présentation avec l'explication du rôle de chaque outil. Nous avons illustré les interfaces graphiques avec une description textuelle

BIBLIOGRAPHIE

[1] : Dantzig G.B., «A Comment on Edie's Traffic Delays at Toll Booths», Operational Research, 2(3) : 339-341, 1954.

[2] : Ariane Partouche, « Planification d'horaires de travail : Méthodologie, Modélisation et résolution à l'aide de la Programmation Linéaire en Nombres Entiers et de la Programmation Par Contraintes », Thèse d'état université PARIS-DAUPHINE, 1998**.**

[3] : Tripathy A. «A Lagrangian Relaxation Approach to Course Scheduling», Journal of the Operational Research Society 31, 1980.

[4] : Jean-Philippe Hamiez et Jin-Kao Hao, « Recherche tabou et planification de Rencontres sportives Tabu Search and sports league scheduling»12èmeCongrès on Reconnaissance des Formes et de l'Intelligence Artificielle –RFIA'2000, Paris, Janvier2000.

[5] : G. M. Thompson. «A simulated-annealing heuristic for shift scheduling using non-continuously available employees» Computers and Operations Research, 23(3)**:**275–288, 1996

[6] :J. C. Régin, «Modeling and solving sports league scheduling with constraint Programming» 1er Congrès de la société Française de Recherche Opérationnelle et Aide à la Décision (ROADEF) Paris, France, 1998.

[7] :CHAN Yew Cheong, Peter, « La planification du personnel : acteurs, actions et termes multiples pour une planification opérationnelle des personnes », Thèse de doctorat, Institut IMAG, Université JOSEPH FOURIER-Greonoble Octobre 2002.

[8] : Scheduling in Sports: An annotated bibliography. Computers and Operations Research, 37, 1-19.

[9] :BOOKTabu Search. Boston: Kluwer.

[10] :A schedule-then-break approach to sports timetabling. International Conference on the Practice and Theory of Automated Timetabling, 242-253.

[11] : BookProgramming with constraints. Cambridge: The MIT Press.

[12] : Un langage et un programme pour énonceretrésoudre des problèmes combinatoires. PhD thesis, Universit´e de Paris VI.

[13] : Colmerauer, A. (1990). An introduction to PROLOG III. Communications of the ACM, 33 :69–90.

[14] :Waltz, D. L. (1975). Understanding line drawings of scenes withShadows. In The Psychology of Computer Vision, pages 19–91. McGraw Hill.D'abord paru dans Tech. Rep AI271, MIT MA, 1972.

[15] :of constraints : FundamentalProperties and applications to picture processing. Information Science, 7 :95–132.

[16] : Consistency in networks of relations.Artificial Intelligence, 8 :99–118.

[17] : Synthesizing constraint expressions. CACM, 21(11) :958–966.

[18] : Caseau, Y., Networks Guillo, P.-Y., and Levenez, E. (1993). A deductiveAnd object-oriented approach to a complex scheduling problem. In Proceedings

Of DOOD'93.

[19] : Van Hentenryck, P., Michel, L., L.Perron, and Régin,J.-C. (1999). Constraint programming in opl. In PPDP 99, International Conférence on the Principles and Practice of Declarative Programming, pages 98–116,Paris, France.

[20] : Beldiceanu, N. and Contejean, E. (1994). Introducing global constraints in chip. Journal of Mathematical and Computer Modelling, 20(12) :97–123.

[21] : Carlier, J. and Pinson, E. (1994). Adjustments of heads

And tails for the jobshop problem. European Journal of Operational Research,

78 :146–161.

[22] : Baptiste, P., Le Pape, C., and Peridy, L. (1998). GlobalConstraints for partial csps : A case-study of resource and due date constraints.In Proceedings CP'98, pages 87–101, Pisa, Italy.

[23] : Zhou, J. (1996). A constraint program for solving the job-shop problem. In Proceedings of CP'96, pages 510–524, Cambridge.

[24] : Zhou, J. (1997). Computing Smallest Cartesian Products of Intervals :Application to the Jobshop Scheduling Problem. PhD thesis, Université de la

Méditerranée, Marseille.

[25] : Simonis, H. (1996). Problem classification scheme for finite domainConstraint solving. In CP96, Workshop on Constraint Programming Applications :An Inventory and Taxonomy, pages 1–26, Cambridge, MA, USA.

[26] : McAloon, K., Tretkoff, C., and Wetzel, G. (1997). SportsLeague scheduling. In Proceedings of ILOG user's conference, Paris

RESUME

Les problèmes de planification d'horaires concernent typiquement les sociétés et services exerçant une activité continue ou quasi-continue. Il en va ainsi par exemple des Compagnies aériennes, Sport, services portuaires, entreprises de transport, hôpitaux,… etc. La planification d'horaires vise à dimensionner la force de travail et à programmer L'utilisation de ressources sur un horizon allant de la journée à quelques mois, de manière à atteindre le meilleur équilibre entre qualité de service, coût et satisfaction sociale. L'objectif de ce mémoire consiste alors à définir et à mettre en œuvre une méthode et Applications permettant de résoudre le problème de planification dans un cas concret : planning d'un Tournoi Sportif utilisant la programmation par contraintes PPC.

ABSTRACT

Scheduling problems typically relate to companies and services engaged in continuous or near-continuous business. This is the case, for example, with airlines, sports, port services, transport companies, hospitals, etc. Scheduling aims at sizing the workforce and programming the use of resources over a day-to-month horizon, in order to achieve the best balance between quality of service, cost and social satisfaction. The objective of this thesis is to define and implement a method and Applications to solve the planning problem in a concrete case: planning a Sports Tournament using CP « constraint programming ».

ملخص

ترتبط مشكلات الجدولة عادةً بالشركات والخدمات التي تعمل في الأعمال المستمرة أو شبه المستمرة. هذا هو الحال ، على سبيل المثال ، مع شركات الطيران والرياضة وخدمات الموانئ وشركات النقل والمستشفيات وغيرها. تهدف الجدولة إلى تحديد حجم القوى العاملة وبرمجة استخدام الموارد على أفق يومي ، من أجل تحقيق أفضل توازن بين جودة الخدمة والتكلفة والرضا الاجتماعي. الهدف من هذه الرسالة هو تحديد وتطبيق طريقة وتطبيقات لحل مشكلة التخطيط في حالة محددة: التخطيط لبطولة رياضية باستخدام البرمجة وفق قيود (أو البرمجة المقيدة بشروط).PPC