Mathematical programming methods for decentralized POMDPs

Raghav Aras

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Mathematical Programming Methods For Decentralized POMDPs

THÈSE

présentée et soutenue publiquement le le 23 octobre, 2008

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Raghav Aras

Composition du jury

Rapporteurs : Nadine Piat, Professeur, Université De Franche-Comté, Besançon
Shlomo Zilberstein, Professeur, University Of Massachussetts, Amherst

Examinateurs : René Schott, Professeur, Université Henri-Poincaré, Nancy
Philippe Mathieu, Professeur, Université De Lille
François Charpillet, Directeur De Recherche, INRIA
Alain Dutech, Chargé De Recherche, INRIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Table of Contents

Table des figures

Liste des tableaux

Première partie

Manuscrit Français

Chapitre 1

Planifier dans les problèmes décentralisés

L'objet de ce chapitre, qui est le pendant du chapitre 1 de la partie anglaise du manuscrit (voir page 55), est de présenter le problème auquel nous allons nous intéresser dans cette thèse. Ce problème est celui de la **planificaiton distribuée dans l'incertain**. Après avoir exposé ce cadre et les problèmes qu'il pose, nous introduirons rapidement plusieurs formalismes dont celui des Processus Décisionnels de Markov Partiellement Observables Décentralisés (DEC-POMDP) qui va particulièrement ous intéresser au cours de cette thèse. Ce chapitre se termine par l'exposé des principales contributions de cette thèse, à savoir l'utilisation d'une formulation différente des DEC-POMDP afin de présenter des algorithmes de résolutions plus rapides et moins gourmand en place mémoire.

1.1 Les problèmes décentralisés

La planification est un concept central à toutes nos activités. Nous faisons tous des plans, que cela soit pour organiser un voyager ou pour gérer le budget de la semaine. Dans l'industrie ou en économie, planifier est fondamental. Dans de nombreuses situations, toutes les données à prendre en compte pour concevoir un plan sont bien trop complexes et inter-dépendantes pour qu'il soit possible de concevoir un plan à la main, et la conception de plan doit ainsi être automatisée. C'est d'autant plus vrai pour les problèmes de décision séquentielle. Ces problèmes sont composés de plusieurs étapes et, à chaque étape, une décision doit être prise en incorporant de nouvelles données.

La complexité computationnelle (en temps et en espace mémoire) de la conception d'un plan dépend de la nature du problème de décision séquentielle. Le paramètre le plus important à prendre en compte à cet égard est l'aspect *centralisé* ou *décentralisé* du problème (de décision séquentielle).

Un plan pour un problème centralisé est exécuté par un seul agent (un agent pouvant être un homme ou une machine). Le résultat du problème dépend seulement des actions de l'agent qui exécute le plan. Un plan pour un problème décentralisé est exécuté par plusieurs agents, chacun étant indépendant des autres (de telle manière qu'un plan décentralisé est en fait un tuple de plans, chaque agent ayant son plan). Ce qu'il advient du problème dépend des plans de *tous* les agents et non des actions d'un seul d'entre eux.

Les deux exemples suivants illustrent la distinction entre les deux types de problèmes.

- (Problème A) : Alice, située au point A, veut se rendre au point C vers midi en partant de A vers 9 heures. Alice veut aussi se rendre au point D vers 10 heures et s'y arrêter un peu avant de continuer vers C.
- (Problème AB) : Alice, située au point A et Bob, qui se trouve en B, veulent tous les deux se retrouver au point C vers midi. Alice partira du point A vers 9 heures et Bob du point B vers 8 heures. Tous les deux voudraient aussi se retrouver au point D vers 10 heures pour quelques minutes avant de se séparer pour se retrouver ensuite au point C.

Le problème A est un problème centralisé. Le fait qu'Alice atteigne ses objectifs (atteindre D et C à l'heure) ne dépend uniquement que de ses actions (si l'on fait abstraction des facteurs extérieurs comme le trafic routier par exemple). Le problème AB est un problème décentralisé. Le fait que Bob et Alice atteignent leurs objectifs communs ne dépend plus des seules actions de l'un d'entre eux mais bien des actions des *deux*.

Concevoir des plans pour des problèmes décentralisés est une tâche beaucoup plus difficile que de concevoir des plans centralisés et ce, pour plusieurs raisons.

Un plan pour un agent, qu'il soit centralisé ou décentralisé, est en fait une liste qui lui prescrit une action pour chaque contingence, ou éventualité, que l'agent peut être amené à rencontrer au cours de l'exécution de son plan. Dans un problème décentralisé, toute action effectuée par un agent détermine non seulement les contingences futures de l'agent mais aussi les contingences futures des autres agents. Les devenirs des agents sont intimement mêlés. L'état du problème, à tout instant, est décrit entre autre par les situations dans lesquelles se trouvent tous les agents à cet instant.

Cependant, dans les problèmes décentralisés, comme les agents exécutent leurs plans de manière indépendante les uns des autres, il en découle implicitement qu'à un moment donné il se peut qu'aucun des agents ne soit en mesure de connaître les contingences des autres agents à cet instant ou aux instants précédents. En général, du fait de cette considération implicite, les agents n'ont pas accès à des informations identiques au cours du déroulement du problème.

Quand on conçoit un plan décentralisé, il faut tenir compte de cet état de fait dont l'implication est qu'une action "optimale" pour un agent qui se trouve dans un état c doit être optimale non seulement pour cet état c mais aussi pour *toutes les contingences* auxquelles pourraient être confrontés les autres agents pendant que l'agent se trouve confronté à c. Cet état de fait résulte en une explosion combinatoire de toutes les situations qu'il faut considérer lors de l'élaboration de plans décentralisés, ce qui en fait une tâche extrêmement compliquée.

Revenons au problème AB. L'état du problème peut être décrit à tout instant par les positions spatiales d'Alice et Bob. Supposons qu'Alice et Bob ne communiquent pas entre eux. Dans ce cas, ni Alice ni Bob n'ont une connaissance complète de l'état du système.

Que doit faire Alice si elle arrive au point D à 10h30 au lieu de 10h, comme prévu ? Doit-elle attendre, en pensant que Bob est en retard lui aussi ? Doit-elle aller directement en C ? Doit-elle retourner en A ? Son plan *doit* être en mesure de lui dire quoi faire dans cette situation.

Mais, pour élaborer une action optimale pour elle dans cette situation, nous devons prendre en compte toutes les positions où peut se trouver Bob à 10h30 mais aussi tous les plans que va suivre Bob à partir de ces positions car Bob et Alice ont toujours comme objectif de se retrouver aussi en C à midi. De ces considérations résulte l'explosion combinatoire que doit prendre en compte tout raisonnement menant à un plan décentralisé.

Notre tâche aurait été simplifiée si nous avions pu assumer qu'Alice et Bob sont capables de communiquer et de s'indiquer leurs positions mutuelles pendant leurs parcours respectifs. Nos difficultés ne viennent pas tant du grand nombre d'endroit où peut se trouver Bob quand il n'est pas en D à 10h30 mais plutôt du fait qu'Alice ne *sache pas* où se trouve Bob, et vice-versa.

Dans le cas d'un plan centralisé comme le problème A, l'état du problème est simplement décrit par la position d'Alice. Son plan est beaucoup plus simple à concevoir. Si elle arrive en D à 10h30 au lieu des 10h prévues, elle n'a pas besoin d'examiner un ensemble de situation démesurées pour décider de quelles actions prendre maintenant car le problème ne dépend que de sa seule situation.

1.2 Exemples

Nous présentons un exemple plus complet et plus compliqué qui servira à illustrer certains de nos propos tout au long de la thèse. D'autres exemples sont présentés dans la partie anglaise du manuscrit (voir sections (1.1.2 du manuscrit anglais) à (1.1.4 du manuscrit anglais)). Ces exemples servirons aussi de *benchmark* pour valider nos algorithmes.

Le problème du canal de broadcast à accès multiple (MABC[1]) [Ros83] est un exemple pratique d'un problème décentralisé. Dans ce problème, il faut décider de comment allouer un unique canal de broadcast à un certain nombre de stations d'émission pour une certaine durée. Chaque station possède un buffer qui peut stocker un certain nombre de messages. Nous faisons l'hypothèse que la durée du problème peut être divisée en périodes discrètes. Durant une période, une seule station peut utiliser le canal pour envoyer un message. Si deux stations émettent un message pendant la même période, il en résulte une collision et les deux messages sont perdus. Un buffer qui est vide ou qui se vide durant une période est rempli avec une certaine probabilité au cours de la prochaine période.

Pour allouer le canal entre les stations, nous devons concevoir une politique d'émission pour chacune des stations. La politique d'émission d'une station détermine si cette station pourra utiliser ou non le canal d'émission pour une période donnée en fonction des informations dont dispose la station à cet instant. Ces informations locales se composent de l'état du buffer de cette station (c'est-à-dire du nombre de messages que contient le buffer). Une station n'a pas accès aux buffers des autres stations. Notre objectif est de formuler les politiques d'émission des stations de manière à ce que le nombre de messages effectivement transmis au travers du canal pendant la durée du problème soit maximal.

[1] de l'anglais "Multi-Access Broadcast Channel

L'objectif n'est pas de construire les politiques les plus "justes" possibles, c'est-à-dire les politiques qui résulteraient en une répartition égale des messages émis par chacune des stations. Ce qui nous intéresse, au travers de l'objectif annoncé, est de maximiser le nombre de messages envoyé, même si ces messages ne sont envoyés que par une seule et même station. Dès lors, concevoir ce genre de politique d'émission serait beaucoup plus simple si chaque station était au courant de l'état des buffers des autres stations. Il suffirait en effet d'assigner un ordre de priorité arbitraire à chaque station et, à un moment donnée, de faire émettre la station dont la priorité est la plus haute parmi les stations dont le buffer n'est pas vide. Le nombre de messages transmis serait alors bien maximal.

Cependant, notre tâche est rendu difficile précisément parce que les stations ne peuvent pas communiquer l'état de leur buffer (ce qui reviendrait à résoudre le problème que nous essayons de résoudre). Comme les besoins des stations varient de manière aléatoire au cours du temps, nous ne pouvons pas non plus concevoir de politique de transmission déterminée à l'avance de manière statique, cela résulterait en un gaspillage de la bande passante. Ainsi, nous devons concevoir des politiques dynamiques qui peuvent même allouer le canal de communication à plusieurs stations pour la même période. Il faut alors prévoir quoi faire en cas de collision.

Quand une collision se produit, les stations qui ont essayé d'émettre peuvent le détecter. Certaines d'entre elles devront alors décider de ne pas envoyer leurs messages de nouveau pour éviter une collision future et de nouveaux messages seront perdus. Mais, cette décision doit aussi tenir compte des états probables des buffers des autres stations, états qui dépendent aussi des allocations précédentes du canal d'émission.

1.3 Modéliser les problèmes décentralisés

Les problèmes décentralisés – sous une forme ou un autre – ont été l'objet d'étude de différentes disciplines comme la théorie des jeux, la recherche opérationnelle, la théorie du contrôle, l'intelligence artificielle, *etc.* Plusieurs modèles mathématiques sont disponibles pour formuler ce genre de problème. H.W. Kuhn a d'abord formalisé les problèmes décentralisés au début des années 1950 comme des jeux sous forme extensive avec information imparfaite [Kuh50]. Le problème de *décision d'équipe*[2], rappelé dans la section (1.1.1), a ensuite été introduit et étudié par Tsitsiklis et Athans [TA85]. Des modèles plus récents pour ce type de problème incluent les diagrammes d'influence multi-agents (ou MAID[3]) [BSK06] et les processus décisionnels de Markov partiellement observables décentralisés (ou DEC-POMDP[4]) [BGIZ02].

Le modèle que nous allons utiliser pour formuler les problèmes décentralisés est celui des DEC-POMDP.

Les DEC-POMDP [BGIZ02] est une généralisation des processus décisionnels de Markov partiellement observables (ou POMDP) [SS73]. Le modèle des POMDP est lui-même une généralisation des processus décisionnels de Markov (MDP) [Bel57].

[2]de l'anglais "team decision problem".
[3]de l'anglais Multi-Agent Influence Diagram.
[4]de l'anglais Decentralized Partially Observable Markov Decision Process.

Le modèle des MDP est une des formulations les plus importantes pour les problèmes centralisés de décision séquentielle. C'est une des pierres de voûte de nombreux travaux en recherche opérationnelle [Put94] et en apprentissage par renforcement [SB98]. En formulant un problème comme un MDP, nous le caractérisons comme un problème de contrôle d'un processus markovien par un agent. De nombreux problèmes, provenant de domaines aussi variés que la gestion de ressources en eau ou que le traitement des réclamations pour les assurances auto, peuvent être résolus en utilisant les MDP [Whi93].

Cependant, un MDP représente un cas idéal du point de vue de l'information dont dispose un agent. L'état du processus y est supposé complètement et totalement observable. Dès lors, l'agent est sensé avoir une connaissance complète du problème. On dit qu'il possède une information parfaite de l'état du système. Mais les problèmes réels imposent des contraintes fortes sur l'information dont disposent les agents qui doivent prendre les décisions. Bien souvent, l'information qui est nécessaire pour prendre une décision optimale n'est que partiellement disponible.

Le modèle des MDP a donc été étendu pour former les POMDP qui permettent de prendre en compte les problèmes centralisés où l'état du processus de Markov n'est que partiellement observable par l'agent. Cette formulation permet de modéliser des problèmes où l'agent n'a qu'une information partielle de l'état. Plusieurs problèmes pratiques intéressants [Cas98b] qui ne rentrent pas dans le cadre des MDP, comme des problèmes de maintenance ou de contrôle qualité [SS73], de sélection de zone de pêche [Lan89], *etc.*, peuvent néanmoins être modélisé et résolus dans le cadre des POMDP.

Le cadre des DEC-POMDP est une généralisation naturelle des POMDP pour les problèmes décentralisés. Un problème modélisé en un DEC-POMDP est caractérisé comme étant un problème de contrôle décentralisé d'un processus markovien par un ensemble d'agents, chacun ayant une vue partielle de l'état. Dans un DEC-POMDP, la connaissance de l'état est d'autant plus partielle que *chaque* agent en a une vue partielle et que cette connaissance distribuée n'est pas suffisante pour reconstituer l'état processus markovien. Ainsi, un DEC-POMDP peut être utilisé pour modéliser des problèmes décentralisés tel que le problème MABC vu précédemment ou le problème de planification pour Alice et Bob (problème AB).

1.4 Contributions de la thèse

L'objet central de cette thèse sont les DEC-POMDP à *horizon fini*. Ces DEC-POMDP modélisent des problèmes décentralisés ayant une durée finie et connue à l'avance. Ainsi, dans la suite de ce manuscrit et à moins qu'il n'en soit fait explicitement mention, les DEC-POMDP considérés seront d'horizon fini. Cette thèse présente de nouveaux algorithmes efficaces pour trouver des politiques optimales jointes pour les DEC-POMDP d'horizon fini.

Bernstein [BGIZ02] a montré qu'en passant du contrôle mono-agent d'un processus de Markov avec des informations partielles au contrôle d'un processus markovien par plusieurs agents, on doit faire face à un accroissement très substantiel en terme de complexité computationnelle. Bernstein *et al.* ont prouvé que résoudre un DEC-POLMDP est un problème bien plus complexe qu'un POMDP. Alors que trouver une politique optimale pour un MDP est un problème P-complet et que trouver une politique optimale pour un POMDP et PSPACE-complet [PT87], le

problème qui consiste à trouver une politique jointe optimale pour un DEC-POMDP est NEXP-dur [BGIZ02].

La grande complexité de résolution des DEC-POMDP fait qu'il n'y a que très peu d'algorithmes exacts pour ces problèmes. Un algorithme exact calcule une politique jointe optimale. Bien que le formalisme des DEC-POMDP soit récent, des formalismes similaires ont été utilisé et étudiés depuis les années 1970 et 1980, notamment dans le domaine de la théorie du contrôle [AM80, Ros83]. Cependant, ces travaux de recherche ne semblent pas avoir abouti à la formulation d'algorithmes génériques pour résoudre les DEC-POMDP. A ce jour, nous ne connaissons que trois algorithmes exacts pour les DEC-POMDP : "Dynamic Programming" [HBZ04], "MAA*" [SCZ05] et "Point Based Dynamic Programming" [SC06].

De manière analogue, jusqu'à très récemment, peu d'algorithmes existent pour résoudre des jeux sous forme extensive avec information imparfaite. Une percée majeure dans le domaine est issue des travaux de D. Koller, B. von Stegel et N. Megiddo qui ont montré, à travers une série de papiers [KMvS94, KM96, vS96], comment de tels jeux pouvaient être efficacement résolus. L'approche qu'ils ont conçu – qui s'appuie sur une utilisation ingénieuse de *politiques exprimées sous forme séquentielle* – a permis de résoudre un jeu en utilisant un espace mémoire qui est linéaire en la taille du jeu ; des approches précédentes nécessitaient un espace mémoire exponentiel en la taille du jeu. Grâce à cette réduction en espace mémoire, des jeux qui étaient hors d'atteinte des algorithmes existants ont pu être résolus et le temps de résolution des jeux qui pouvaient l'être à été significativement réduit. L'approche décrite par Koller *et al.* apportait en fait des *algorithmes rapides* pour les jeux sous forme extensives avec information imparfaite.

Or, un DEC-POMDP avec horizon fini peut être vu comme un jeu sous forme extensive avec information imparfaite. Autrement dit, résoudre un DEC-POMDP avec horizon fini pose les mêmes problèmes que la résolution des jeux sous forme extensive avec information imparfaite et récompenses identiques. Cependant, bien que l'approche de Koller *et al.* précède les travaux sur la résolution des DEC-POMDP, cette approche n'a pourtant pas attiré l'attention lors des recherches sur la résolution des DEC-POMDP.

La contribution de cette thèse est d'avoir adapté l'approche de Koller *et al.* en s'appuyant sur une formulation séquentielle à la résolution des DEC-POMDP. L'adaptation n'est pas simple pour plusieurs raisons. L'approche de Koller *et al.* consiste à formuler un jeu comme un problème linéaire complémentaire (LCP[5]) [Mur88]. Un LCP peut être résolu en utilisant l'algorithme de pivot complémentaire de Lemke [Lem65]. Une solution d'un LCP est un équilibre de Nash du jeu en question. L'adaptation de cette méthode pose plusieurs défis.

1. Comme l'approche de Koller *et al.* a pour but de résoudre des jeux, elle ne cherche qu'un équilibre de Nash du jeu. Un équilibre de Nash d'un DEC-POMDP est une politique jointe *localement* optimale. Un équilibre de Nash n'est pas une solution satisfaisante à un DEC-POMDP car la différence entre la valeur d'une politique optimale jointe et d'une politique jointe localement optimale peut être arbitrairement grande. Pour résumer, trouver une politique jointe localement optimale n'est pas beaucoup mieux que choisir une politique jointe au hasard.

[5] de l'anglais "Linear Complementary Problem".

2. L'approche de Koller *et al.* est dimensionnée pour les jeux à deux joueurs, ce qui veut dire que, même en admettant qu'on puisse l'adapter directement, elle ne pourrait être utilisée que pour les DEC-POMDP avec deux agents.

Cette thèse affronte ces deux défis en proposant de nouveaux programmes mathématiques. Chacun de ces programmes est un programme linéaire mixte 0-1 entier, de la famille des programmes linéaires. Les solutions de ces programmes mathématiques sont des politiques jointes optimales. Les différents programmes ont été élaborés en utilisant différentes propriétés des DEC-POMDP et ont des performances différentes, ainsi que nous le montrerons sur quelques exemples. Ces programmes ont comme point commun qu'ils sont des algorithmes *rapide* pour les DEC-POMDP, tout comme les algorithmes de Koller *et al.* étaient des algorithmes *rapides* pour les jeux à deux joueurs. Le temps de calcul pour résoudre un DEC-POMDP avec nos programmes mathématiques est plus court, d'un ordre de magnitude de 1 ou 2 (en fonction des problèmes) que le temps de calcul des algorithmes existants.

L'autre contribution importante de cette thèse est l'introduction d'heuristiques, inspirées par celles qui sont utilisées par les algorithmes existants, pour accélérer encore les temps de résolution de nos programmes mathématiques. Ces heuristiques profitent de la compacité des DEC-POMDP et nous permettent de réduire la taille des programmes mathématiques, et donc les temps de résolution.

1.5 Plan de la thèse

La partie française du manuscrit suit l'organisation de la partie anglais de ce dernier.

Les chapitres 2 et 3 mettent en place les préliminaires. Le chapitre 2 présente le modèle DEC-POMDP à horizon fini ainsi que les concepts de politique, de politique jointe et de valeurs de ces politiques. Il se termine par un tour d'horizon des algorithmes existants. Le chapitre 3 décrit un concept majeur de cette thèse : la **formulation séquentielle d'une politique** ainsi que la valeur d'une politique sous forme séquentielle. Nous y présentons aussi un premier programme mathématique qui inmplémente les contraintes que doit respecter une politique sous forme séquentielle, ce qui permet de proposer un programme mathématique **non-linéaire** pour résoudre un DEC-POMDP.

Les chapitres 4 et 5 composent le cœur de la thèse et ont pour but de linéariser le programme mathématique précedent afin de pouvoir le résoudre. Dans le chapitre 4, nous montrons comment des considérations combinatoriales nous permettent de proposer deux programmes linéaires mixtes entiers à valeur dans 0-1 pour résoudre des DEC-POMDP. Le chapitre 5 exploite des concepts issus de la théorie des jeux sous forme extensive pour proposer deux autres programme linéaires mixtes entiers plus performants, en s'appuyant notamment sur des travaux de Koller, von Stengel et Megiddo.

Au chapitre 6, nous présentons des heuristiques inspirées de la programmation dynamique pour accélérer la résolution des programmes mathématiques précédents en améliorant leur besoin en place mémoire. Ces algorithmes sont testés expérimentalement et les résultats de ces tests sont présentés dans le chapitre 7.

Les annexes, qui ne font pas partie de la partie française du document, présentent des extensions immédiates de nos algorihmes pour résoudre des *jeux stochastiques partiellement observables*, domaine où de tels algorithmes sont rares.

Chapitre 2

Contrôle décentralisé d'un processus de Markov

Ce chapitre offre une vue synthétique du chapitre 2 de la version anglaise de ce document, que l'on trouvera à la page 67. Le but est de présenter le formalisme des Processus Décisionnels de Markov Partiellement Observables Décentralisé (DEC-POMDP) et quelques concepts clefs qui leur sont associés. Nous y parlerons entre autre de politique, de politique jointe et de leurs fonctions valeur. Enfin, nous donnerons un aperçu des principaux algorithmes existants pour résoudre des DEC-POMDP, que cela soit de manière exacte ou approchée.

2.1 Le modèle DEC-POMDP

Le modèle formel d'un Processus Décisionnel de Markov Partiellement Observé Décentralisé (DEC-POMDP) d'horizon fini est donné par les élements suivants :

- Un ensemble $I = \{1, 2, \ldots, n\}$ de $n \geq 2$ agents.
- Un ensemble S d'états. L'ensemble des distributions de probabilité sur S est noté $\Delta(S)$.
- Pour chaque agent $i \in I$, un ensemble A_i d'actions. L'ensemble $\times_{i \in I} A_i$ est noté A et est appelé l'ensemble des actions jointes. Dans une action jointe a, l'action de l'agent i est notée a_i.
- Pour chaque agent $i \in I$, un ensemble O_i d'observations. L'ensemble $\times_{i \in I} O_i$ est noté O et est appelé l'ensemble des observations jointes. Dans une observations jointe o, l'observation de l'agent i est notée o_i.
- Une fonction de transition $\mathbb{P}$. Pour chaque $s, s' \in S$ et pour chaque $a \in A$, $\mathbb{P}(s, a, s')$ est la probabilité que l'état du problème dans la période actuelle soit s' si, dans la période précédente, l'état était s et si les agents ont exécuté l'action jointe a.
- Une fonction d'observation jointe $\mathbb{G}$. Pour chaque $a \in A$, pour chaque $o \in O$ et pour chaque $s \in S$, $\mathbb{G}(a, s, o)$ est la probabilité que les agents reçoivent l'observation jointe o (c'est-à-dire que chaque agent i reçoive l'observation o_i) si l'état du problème dans cette période est s et si les agents ont exécuté l'action jointe a.
- Une fonction de récompense R. Pour chaque $s \in S$ et pour chaque $a \in A$, $R(s, a) \in \mathbb{R}$ est la récompense obtenue par les agents s'ils ont effectué l'action jointe a quand l'état du processus était s.
- Un état initial $\alpha \in \Delta(S)$. Pour chaque $s \in S$, $\alpha(s)$ est la probabilité que l'état du système dans la première période est s.
- Un entier $T \geq 1$ qui est appelé l'horizon du problème. La durée du problème est de T

périodes temporelles.

Les éléments S, A et $\mathbb{P}$ définissent un processus de Markov. R spécifie ce que l'on attend du contrôle et $\mathbb{G}$ la manière dont l'état est observable par les agents.

L'évolution du processus est gouverné par les actions des agents. Si, à la période t, le système est dans l'état $s^t \in S$, le système va transiter vers un nouvel état $s^{t+1} \in S$ en fonction de l'action jointe des agents a^t avec une probabilité $\mathbb{P}(s^t, a^t, s^{t+1})$. Chaque agent, en fonction de sa vision partielle o_i^{t+1} et de la récompense reçue r^{t+1}, va choisir sa prochaine action a_i^{t+1}, formant ainsi l'action jointe a^{t+1}. Et ainsi de suite.

Trois hypothèses importantes sont faites : (I) à chaque période, les agents ne peuvent connaître l'état s^t du système. (II) à chaque période, un agent ne connaît pas les actions ou les récompenses des autres agents. (III) à chaque période, un agent connaît les séquences d'actions qu'il a effectuées jusqu'à présent ainsi que les observaions qu'il a reçues, on parle alors de *rappel parfait*[6].

L'objectif des agents est de contrôler le processus Markovien de manière à maximiser l'espérance de la somme des récompenses reçues

$$\sum_{t=1}^{T} R(s^t, (a_1^t, a_2^t, \ldots, a_n^t))$$

.

2.2 Politique et politique jointe

Une politique d'un agent est un plan d'action complet pour les T étapes du DEC-POMDP. Elle doit indiquer, pour chaque situation que peut rencontrer l'agent, une action à effectuer. Dans le cas d'un DEC-POMDP, il faut donc indiquer une action pour chaque séquence d'observations possibles.

Formellement, notons $\overline{O}_i^k$ l'ensemble des séquences de k observations pour l'agent $i \in I$. $\overline{O}_i^0$ ne contient que l'observation nulle $\varnothing$. Ainsi, une **politique** d'horizon t pour un agent i est une fonction π qui, pour chaque entier k de $0, \cdots, t-1$, pour chaque séquence d'observation $\overline{o} \in \overline{O}_i^k$ associe une action $\pi(\overline{o}) \in A_i$. L'ensemble des politiques d'horizon t d'un agent i est noté Π_i^t.

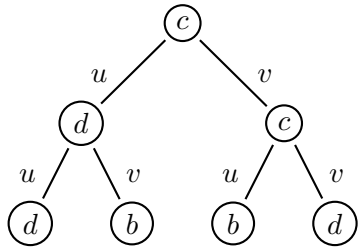
Les politiques de Π_i^t, qui sont déterministes, sont dite *pures*. Une politique *mixte* d'un agent i est donnée par une distribution de probabilité δ sur $\Delta(\Pi_i^t)$. Avant le début d'un épisode, l'agent choisi donc une politique π avec une probabilité $\delta(\pi)$ et applique cette politique pour tout l'épisode d'horizon T.

Le tableau suivant donne un exemple de politique d'horizon 3 pour un agent dont l'ensemble d'obervation est $O_i = \{u, v\}$ et l'ensembles des actions est $A_i = \{a, b, c, d\}$.

Séquence d'observation	$\varnothing$	u	v	uu	uv	vu	vv
Action à choisir	c	d	c	d	b	b	d

[6] en anglais, "perfect recall".

Cette politique est classiquement représentée par un arbre où les noeuds représentent les actions à effectuer quand on arrive à ce noeud après avoir reçu la séquence d'observations correspondant aux arcs traversés pour arriver au noeud en partant de la racine de l'arbre. Chaque arc est etiquetté par une observation. Par exemple, pour la politique précédente, on obtiendrait :



La politique composée des politiques de tous les agents forme une politique jointe. Formellement, une **politique jointe** d'horison t $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ est un tuple de n politiques d'horizon t où, pour chaque $i \in I$, σ_i est la politique d'horizon t de l'agent i. La politique de l'agent i au sein de la politique jointe σ est notée σ_i. L'ensemble $\times_{i\in I}\Pi_i^t$ des politiques jointes d'horizon t est noté Π^t.

Pour un agent $i \in I$, une **politique jointe i-réduite** σ' est le tuple formé par les $n-1$ politiques d'horizon t des $n-1$ autres agents. L'ensemble $\times_{k\in I\setminus\{i\}}\Pi_k^t$ des politiques i-réduite de l'agent i est noté Π_{-i}^t. La politique jointe (π, σ') est la politique jointe formée de la politique π de l'agent i et d'une politique i-réduite σ' de Π_{-i}^t.

2.2.1 Valeur d'une politique jointe

La valeur d'une politique jointe σ est l'espérance des récompenses accumulées par les agents durant les T périodes du DEC-POMDP. Nous noterons cette valeur $V(\alpha, \sigma)$. Ainsi que nous le détaillons plus amplement à la section (2.1.1 du manuscrit anglais), cette valeur se définit de manière récursive par :

$$V(\alpha, \sigma) \quad = \quad V(\alpha, \sigma, \varnothing) \tag{2.1}$$

avec les éléments de récursion suivants :

1.
$$V(\alpha, \sigma, \varnothing) \quad = \quad R(\alpha, \sigma(\varnothing)) + \sum_{o\in O} \mathcal{T}(o|\alpha, \sigma(\varnothing))V(\alpha', \sigma, o) \tag{2.2}$$

   (α' est l'état du processus obtenu en appliquant a dans l'état α).

2. Pour chaque $\beta \in \Delta(S)$, pour chaque k de $\{1, \cdots, t-2\}$, pour chaque $\overline{o} \in \overline{O}^k$,
$$V(\beta, \sigma, \overline{o}) \quad = \quad R(\beta, \sigma(\overline{o})) + \sum_{o\in O} \mathcal{T}(o|\beta, \sigma(\overline{o}))V(\beta', \sigma, \overline{o}o) \tag{2.3}$$

   (β' est la distribution sur les états obtenue en appliquant $\sigma(\overline{o})$ à partir de β et o. $\overline{o}o$ est la séquence d'observations jointe obtenue en concaténant o à $\overline{o}$.)

3. Pour chaque $\beta \in \Delta(S)$, pour chaque $\overline{o} \in \overline{O}^{t-1}$,
$$V(\beta, \sigma, \overline{o}) \quad = \quad R(\beta, \sigma(\overline{o})) \tag{2.4}$$

$\mathcal{T}(o|\beta, a)$ est la probabilité que les agents observent o si, en fonction de la probabilité β sur S, on exécute l'action jointe a.

$$\beta'(s') \quad = \quad \frac{\sum_{s \in S} \beta(s)\mathbb{P}(s, a, s')\mathbb{G}(a, s', o)}{\mathcal{T}(o|\beta, a)} \tag{2.5}$$

$R(\beta, a)$ est l'espérance de la récompense reçue par les agents si la distribution sur les états est β. Ces valeurs sont précisées en section 2.1.1 du manuscrit anglais, page 69.

2.2.2 Politique optimale jointe

Une **politique jointe optimale** est une politique jointe dont la valeur est maximale, c'est-à-dire une politique σ^* de Π^T telle que :

$$\sigma^* \quad = \quad \arg\max_{\sigma \in \Pi^T} V(\alpha, \sigma). \tag{2.6}$$

Une politique d'horizon T est dite **localement optimale** en α si elle vérifie :

$$V(\alpha, \sigma) \quad \geq \quad V(\alpha, (\pi, \sigma_{-i})), \quad \forall i \in I, \forall \pi \in \Pi_i^T. \tag{2.7}$$

Les politiques localement optimales sont des équilibres de Nash du jeu formé par le DEC-POMDP au sens où, individuellement, aucun agent n'a intérêt à changer sa politique. Mais si toute politique jointe optimale est aussi localement optimale, l'inverse n'est pas vrai.

En annexe (A), nous présentons une autre forme de solution pour les DEC-POMDP en s'appuyant sur les *équilibres corrélés*. Nous y proposons un algorithme pour trouver ce type de solution qui, pour les jeux, a été étudié récemment par Von Stengel et Forges [vSF06].

2.3 Algorithmes existants

Une solution pour trouver une politique jointe optimale d'horizon T est d'énumérer toutes les politiques et de choisir la meilleure. C'est une solution qui devient rapidement impossible à mettre en œuvre car le nombre de politiques est doublement exponnentiel en le nombre d'observations et en la taille du problème. Ainsi, le nombre de politiques d'horizon t pour un agent i est $|A_i|^{\frac{|O_i|^t - 1}{|O_i| - 1}}$.

Les algorithmes existants peuvent être classés en trois groupes : exacts (E), localement exacts (LE) et approchés (A) pour ceux qui cherchent une solution approchée au problème. Le tableau suivant liste les différents algorithmes existants à ce jour, les détails sur ces algorithmes étant donnés dans la partie anglaise du manuscrit (voir sections (2.2 du manuscrit anglais) et (2.3 du manuscrit anglais)).

Algorithme	Réf.	Type
Coevolution	[CSC02]	LE
Joint Equilibrium Search For Policies (JESP)	[NTY+03]	LE
Dynamic Programming (DP)	[HBZ04]	E
Multi Agent A* (MAA*)	[SCZ05]	E
Continuous Space JESP	[VNTY06]	LE
Point Based Dynamic Programming (PBDP)	[SC06]	E
Memory Bounded Dynamic Programming (MBDP)	[SZ07]	A
Generalized MAA*	[OSV08]	E

2.4 Bilan

Ce chapitre a permi de présenter le problème formel qui nous intéresse ainsi que l'état de l'art concernant sa résolution. En s'appuyant principalement sur la programmation dynamique, les algorithmes actuels ont une complexité telle que nous avons voulu approcher ce problème sous un autre angle : celui des "programmes mathématiques". Cette étude fait l'objet des chapitres suivants.

Chapitre 3

Politique en formulation séquentielle

Ce chapitre, qui est le pendant du chapitre 3 se trouvant à la page 83, propose une formulation différente d'un DEC-POMDP. Notre approche s'inspirant des travaux de D. Koller, B. von Stengel et N. Megiddo sur la résolution de jeux mathématiques sous-forme extensive [KMvS94, KM96, vS96]. Une politique est décrite par l'ensemble des trajectoire qu'elle peut générer et que nous appelerons des *historiques*. En associant un *poids* à ces historiques, nous cherchons les poids qui, tout en respectant les contraintes assurant qu'ils définissent bien une politique, permettent de maximiser la fonction valeur de cette politique.

Comme il ne faut qu'un nombre exponentiel d'historiques pour définir une politique (et non doublement exponentiel), les algoritmes que nous allons proposer en s'appuyant sur cette approche sont plus efficaces. Cette idée sera expérimentalement validée sur plusieurs exemples typiques de DEC-POMDP.

3.1 Historique et ensemble d'information

Soit un DEC-POMDP (I, S, $\{A_i\}$, $\{O_i\}$, R, $\mathbb{P}$, $\mathbb{G}$, T), comme nous l'avons décrit au Chapitre 2.

Un **historique** h d'un agent $i \in I$ est constitué d'une suite (a^1, o^1, a^2, o^2, …, o^t,a^{t+1}) de t observations et $t+1$ actions. La longueur d'un historique est le nombre d'actions qu'il contient. Pour un historique h, nous noterons $a^k(h)$ la k-ème action et $o^k(h)$ la k-ème observation. Un historique de longueur T est un **historique terminal**. L'historique de longueur nulle est noté $\varnothing$.

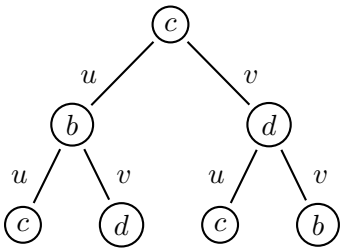
Pour un agent, $i \in I$, l'ensemble des historiques de taille t est noté $\mathcal{H}_i^t$. $\mathcal{H}_i$ est l'ensemble de tous les historiques de l'agent i dont la taille est inférieure ou égale à T. La taille de $\mathcal{H}_i$ vaut donc :

$$n_i \quad = |\mathcal{H}_i| \quad = \sum_{t=1}^{T} |A_i|^t |O_i|^{t-1}. \tag{3.1}$$

L'ensemble $\mathcal{H}_i^T$ des historiques terminaux est noté $\mathcal{E}_i$. Et nous noterons $\mathcal{N}_i$ l'ensemble $\mathcal{H}_i \backslash \mathcal{H}_i^T$ des historiques non-terminaux.

Un **ensemble d'information** ι d'un agent $i \in I$ est une suite $(a^1, o^1, a^2, o^2, \ldots, o^t)$. La longueur d'un ensemble d'information est le nombre d'action qu'il contient. Un ensemble d'information de longueur $T-1$ est un **ensemble d'information terminal**. Pour un historique h, on note $\iota(h)$ l'ensemble d'information constitué en enlevant de h sa dernière action.

Considérons, pour un agent, la politique pure suivante :



Cette politique, où b, c et d sont des actions, u et v des observations, permet de générer les historiques suivants : (c), (cub), (cvd), $(cubuc)$, $(cubvd)$, $(cvduc)$ et $(cvdvb)$. Il suffit d'enlever à chacun de ces historiques sa dernière action pour obtenir les états d'information $(\varnothing)$, (cu), (cv), $(cubu)$, $(cubv)$, $(cvdu)$ et $(cvdv)$.

3.1.1 Politique sous forme séquentielle

Une **fonction d'historique** θ d'un agent i est une fonction qui assigne à chaque historique h de $\mathcal{H}_i$ un nombre $\theta(h)$ dans l'intervalle $[0;1]$. Ce nombre $\theta(h)$ est appellé le **poids** de h dans θ.

Pour une politique π donnée, il est assez simple de construire la fonction d'historique associée en écrivant que :

$$\theta(h) = \Pr_{\pi}\{a^{|h|}|\iota(h)\} \qquad \forall h \in \mathcal{H}_i$$

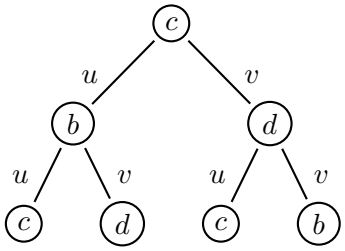
Par contre, toute fonction d'historique ne définit pas forcément une politique. Il faut pour cela que certaines conditions soient réunies. Ainsi, pour qu'une fonction d'historique p définisse une **politique sous forme séquentielle** d'horizon T, les conditions suivantes doivent être vérifiées :

$$\sum_{a \in A_i} p(a) = 1 \tag{3.2}$$

$$-p(h) + \sum_{a \in A_i} p(hoa) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{3.3}$$

où hoa est l'historique obtenu en concaténant o et a à h. L'ensemble des politiques sous forme séquentielle est noté X_i et nous appellerons $\hat{X}_i \subset X_i$ l'ensemble des politiques séquentielles pures.

L'exemple ci-dessous présente une politique pure et la fonction d'historique associée.



history	θ
c	1
cub	1
cvd	1
$cubuc$	1
$cubvd$	1
$cvduc$	1
$cvdvb$	1

L'idée de représenter une politique par l'ensemble des historiques qu'elle génère est inspirée des travaux de Koller et Von Stengel sur les jeux en forme extensive [KMvS94, vS96]. La partie anglaise du manuscrit donne aussi des exemples de politique stochastique sous forme séquentielle (voir section 3.2 du manuscrit anglais, page 84).

3.2 Contraintes de politique et programmation linéaire

On peut trouver une fonction d'historique vérifiant les contraintes de politique précédentes en utilisant la programmation linéaire. De fait, le programme linéaire suivant implémente la définition d'une politique sous forme séquentielle.

Contraintes de Politique

$$\sum_{a\in A_i} x(a) = 1 \tag{3.4}$$

$$-x(h) + \sum_{a\in A_i} x(hoa) = 0, \quad \forall h \in \mathcal{N}_i \backslash H_i^T, \forall o \in O_i \tag{3.5}$$

$$x(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{3.6}$$

Les variables $x(h)$ de ce programme linéaire représentent les poids des historiques dans la politique. Ce programme linéaire est exponentiel en la taille de l'horizon, c'est-à-dire que le nombre de variables et de containtes est expontiel en T.

La seule contrainte sur les variables est qu'elles doivent être positives (éq. 3.6). Mais, ainsi que le montrent le lemme ci-dessous, dont la démonstration se trouve dans la partie du manuscrit en langue anglaise, les contraintes du programme linéaire sont telles que ces variables ne prendront des valeurs que dans l'intervalle $[0; 1]$.

Lemme 3.3. *Dans toute solution x^* de (3.4)-(3.6), $x^*(h) \in [0, 1]$ pour chaque $h \in \mathcal{H}_i$. (voir démonstration page 90).*

Dans le cas où l'on veut se restreindre aux politiques pures, un deuxième lemme permet de n'imposer qu'aux variable x liée à un historique terminal de ne prendre leur valeur que dans l'ensemble $\{0; 1\}$. Ce sont les contraintes du programme linéaire qui propageront ces contraintes aux autres variables.

Lemme 3.4. *Si dans (3.4)-(3.6), la contrainte (3.6) est remplacée par,*

$$x(h) \geq 0, \quad \forall h \in \mathcal{N}_i \tag{3.7}$$

$$x(h) \in \{0, 1\}, \quad \forall h \in \mathcal{H}_i^T \tag{3.8}$$

alors, dans chaque solution x^* du programme linéaire mixte entier résultant, $x^*(h)$ vaut 0 ou 1 pour chaque $h \in \mathcal{H}_i$. (voir démonstration page 90).

Par la suite nous utiliserons de préférence une écriture matricielle du programme linéaire en synthétisant les contraintes (3.4) et (3.5) en une seule matrice de contraintes. La matrice C_i est une matrice creuse.

$$C_i x_i = c_i \tag{3.9}$$
$$x_i \geq 0 \tag{3.10}$$

Si l'on prend des ensembles d'actions et d'observations définis par $A_i = \{b, c\}$ et $O_i = \{u, v\}$, toutes les politiques possibles d'un agents sont ainsi solution du programme linéaire suivant :

$$\begin{aligned} x_i(b) + x_i(c) &= 1 \\ -x_i(b) + x_i(bub) + x_i(buc) &= 0 \\ -x_i(c) + x_i(cub) + x_i(cuc) &= 0 \\ -x_i(b) + x_i(bvb) + x_i(bvc) &= 0 \\ -x_i(c) + x_i(cvb) + x_i(cvc) &= 0 \end{aligned}$$

ce qui se traduit avec une écriture matricielle par :

$$C_i = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} c_i = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

3.3 Formulation séquentielle d'un DEC-POMDP

Les définition et les formulations précédentes, qui étaient données pour un agent, s'étendent naturellement à plusieurs agents et donc à un DEC-POMDP. Nous pouvons utiliser les notions d'**historique joint** et de **politique jointe sous forme séquentielle** en considérant des vecteurs d'historique et de fonction d'historique. Quant au poid d'un historique joint j pour la politique jointe $p = (p_1, p_2, \ldots, p_n)$, nous le définissons comme étant le produit des poids des historiques $j_1,\ldots,j_n$, c'est-à-dire que $p(j) = \prod_{i \in I} p_i(j_i)$.

Nous sommes maintenant à même de proposer une reformulation du problème des DEC-POMDP en utilisant la formulation sous forme de séquence. Ce n'est qu'une reformulation. Etant donné un DEC-POMDP , la formulation de ce même DEC-POMDP est caractérisée par le tuple $(I, \{\mathcal{H}_i\}, \Psi, \overline{R})$ où :

- $I = \{1, 2, \ldots, n\}$ est l'ensemble des agents.
- Pour chaque agent $i \in I$, $\mathcal{H}_i$ est l'ensemble des historiques de taille inférieure ou égale à l'horizon T de l'agent, ainsi que nous l'avons défini à la section 3.1. Chaque ensemble $\mathcal{H}_i$ est construit en utilisant les ensembles A_i et O_i.
- Ψ est la fonction de probabilité conditionnelle sur les historiques joints. Pour chaque historique joint $j \in \mathcal{H}$, $\Psi(\alpha, j)$ est la probabilité d'occurrence de j conditionnée par le fait que les agents exécutent les actions jointes définies par cet historique en sachant que l'état

initial du DEC-POMDP est α. Cette fonction est construite à partir de l'ensemble des états S, de la fonction de transition jointe $\mathbb{P}$ et de la fonction d'observation jointe $\mathbb{G}$.
– $\overline{V}$ est la fonction de valeur jointe. Pour chaque historique joint $j \in \mathcal{H}$, $\overline{R}(\alpha, j)$ est la valeur (au sens de la récompense espérée) que l'agent obtient si l'historique j est effectivement réalisé. Cette fonction est construite à partir de l'ensemble des états S, de la fonction de transition jointe $\mathbb{P}$, de la fonction d'observation jointe $\mathbb{G}$ et de la fonction de récompense R. $\overline{V}$ peut aussi être une fonction qui dépend de Ψ et de R.

La section 3.5 du manuscrit anglais, page 91 détaille comment les fonctions Ψ et $\overline{V}$ peuvent être calculées. Ψ, la probabilité conditionnelle d'un historique joint, vaut :

$$\Psi(\alpha, j) \quad = \quad \text{Prob.}(o^1(j), o^2(j), \ldots, o^{t-1}(j) | \alpha, a^1(j), a^2(j), a^{t-1}(j)). \tag{3.11}$$

On peut exprimer cette probabilité à partir de la probabilité d'obtenir une observation jointe o après avoir exécuté une action a alors que le *belief state* sur les états du processus est β. Cette probabilité $\mathcal{T}(o|\beta, a)$ vaut :

$$\mathcal{T}(o|\beta, a) \quad = \quad \sum_{s \in S} \beta(s) \sum_{s' \in S} \mathbb{P}(s, a, s') \mathbb{G}(a, s', o). \tag{3.12}$$

Comme il est possible, le long d'un historique joint j donné, de calculer itérativement les *belief state* en partant de l'état estimé initial donné par $\beta_j^0 = \alpha$ en utilisant, pout tout k de $\{1, \ldots, t-1\}$:

$$\beta_j^k(s') \quad = \quad \frac{\sum_{s \in S} \beta(s) \mathbb{P}(s, a^k(j), s') \mathbb{G}(a, s', o^k(j))}{\mathcal{T}(o^k(j) | \beta_j^{k-1}, a^k(j))}, \quad \forall s' \in S \tag{3.13}$$

on obtient finalement que :

$$\Psi(\alpha, j) \quad = \quad \prod_{k=1}^{t-1} \mathcal{T}(o^k(j) | \beta_j^{k-1}, a^k(j)). \tag{3.14}$$

Quant à la **valeur d'un historique joint**, on peut l'exprimer par :

$$\overline{V}(\alpha, j) \quad = \quad \overline{S}(\alpha, j) \Psi(\alpha, j) \tag{3.15}$$

où :

$$\overline{S}(\alpha, j) \quad = \quad \sum_{k=1}^{t} \sum_{s \in S} \beta_j^{k-1}(s) R(s, a^k(j)). \tag{3.16}$$

Ce qui permet finalement d'exprimer $\mathcal{V}(\alpha, p)$, la **valeur d'une politique jointe exprimée sous forme séquentielle** p comme étant la somme pondérée de la valeur de ses historiques joints, c'est-à-dire :

$$\mathcal{V}(\alpha, p) \quad = \quad \sum_{j \in H^t} p(j) \overline{V}(\alpha, j) \tag{3.17}$$

où $p(j) = \prod_{i \in I} p_i(j_i)$.

3.4 Programmation mathématique et DEC-POMDP

En utilisant une formulation séquentielle d'un DEC-POMDP, nous savons exprimer les contraintes de politiques sous la forme de programme linéaire et la valeur d'une politique jointe comme une combinaison de la valeur de chaque historique. Dès lors, il est possible de proposer de résoudre un DEC-POMDP à l'aide d'un programme non-linéaire (NLP) de la manière suivante :

Programme Non-Linéaire

$$\text{maximiser} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{3.18}$$

en tenant compte des contraintes de politique,

$$C_i x_i = c_i, \quad \forall i \in I \tag{3.19}$$

$$x_i \geq 0, \quad \forall i \in I \tag{3.20}$$

Résoudre ce genre de problème est complexe, ne serait-ce que parce que la fonction objectif n'est pas convexe. Une solution potentielle serait de rechercher la valeur de cette fonction à chaque sommet de l'ensemble des contraintes mais ces points correspondent à l'ensemble de toutes les politiques jointes pures, ensemble dont la taille est doublement exponentielle en l'horizon du problème.

Les chapitres suivants vont donc proposer des solutions plus efficaces pour permettre de résoudre ce programme non-linéaire sans avoir à énumérer toutes les possibilités.

Chapitre 4

Une approche d'optimisation combinatoriale

Le principale problème posé par le programme mathématique non-linéaire général présenté précédemment est sa fonction objectif qui n'est pas linéaire. Ce chapitre, qui synthétise le chapitre 4 de la version anglaise du document que l'on trouvera à la page 97, s'attaque à ce problème en linéarisant la fonction objectif. Nous verrons que cela nous oblige à restreindre l'espace des solutions

4.1 Linéarisation du problème

Le principe que nous avons suivi pour linéariser le problème est finalement assez simple et s'appuie sur l'utilisation de variables $z(j)$ représentant le produit des variables $x_i(j)$, ce qui permet, dans la fonction objectif, de remplacer le produit $\prod_{i\in I} x_i(j_i$ par une seule variable. Ainsi, la fonction objectif qui était :

$$\text{maximize} \quad \sum_{j\in\mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i\in I} x_i(j_i) \tag{4.1}$$

devient

$$\text{maximize} \quad \sum_{j\in\mathcal{E}} \mathcal{R}(\alpha, j) z(j). \tag{4.2}$$

Il faut néanmoins s'assurer qu'il existe une bijection entre les variables z et les produits $\prod_{i\in I} x_i$. Pour ce faire, nous devons nous restreindre et ne rechercher que des politiques pures, ce qui permet toujours de résoudre le problème puisqu'il admet une politique jointe optimale qui est pure. Dans ce cadre, toutes nos variables prennent leurs valeurs dans $\{0; 1\}$ et l'équivalence entre les variables z et x s'exprime comme suit :

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I. \tag{4.3}$$

Comme chaque agent doit avoir $|O_i|^{T-1}$ historiques terminaux, deux contraintes permettent d'exprimer ce que nous voulons garantir. D'une part, pour assurer que $z(j)$ ne vaut 1 que si suffisamment de variables x_i valent aussi 1, nous écrivons que :

$$\sum_{i=1}^{n} x_i(j_i) - nz(j) \quad \geq \quad 0, \quad \forall j \in \mathcal{E}. \tag{4.4}$$

D'autre part, pour restreindre le nombre de variables z valant 1, nous énumerons le nombre d'historiques terminaux joints et écrivons que :

$$\sum_{j \in H^T} z(j) \quad = \quad \prod_{i \in I} |O_i|^{T-1} \tag{4.5}$$

.

Ainsi, on peut remplacer le programme non-linéaire (3.18)-(3.20) par un programme linéaire suivant, où toutes les variables sont entière et prennent leurs valeurs dans $\{0; 1\}$.

Programme linéaire entier

$$\text{maximiser} \qquad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.6}$$

sous les contraintes,

$$C_i x_i \quad = \quad c_i, \quad \forall i \in I \tag{4.7}$$

$$\sum_{j \in H^T} z(j) \quad = \quad \prod_{i \in I} |O_i|^{T-1} \tag{4.8}$$

$$\sum_{i=1}^{n} x_i(j_i) - n z(j) \quad \geq \quad 0, \quad \forall j \in \mathcal{E} \tag{4.9}$$

$$x_i(h) \quad \in \quad \{0, 1\}, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{4.10}$$

$$z(j) \quad \in \quad \{0, 1\}, \quad \forall j \in \mathcal{E} \tag{4.11}$$

De plus amples détails sur ce programme linéaire sont donnés section 4.2 du manuscrit anglais, page 100. On y trouve en particulier la démonstration du lemme suivant :

Lemme 4.1. *Toute solution (x^*, z^*) de (4.6)-(4.11) satistait la condition suivante pour chaque historique terminal joint $j \in H^T$,*

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.12}$$

Ce lemme, démontré à la page 101, permet de démontrer que ce programme linéaire répond bien à nos attente par le biais du théorème suivant :

Théorème 4.1. *Une solution (x^*, z^*) du programme linéaire (4.6)-(4.11) permet de définir une politique jointe optimale $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$.*

La preuve de ce théorème se trouve page 102.

La linéarisation ainsi proposée augmente considérablement le nombre de variables et de contraintes du programme mathématique à résoudre puisqu'il est maintenant exponentiel en T *et* en n, le nombre d'agents. La section qui suit présente une autre linéarisaion du problème, moins gourmande en taille.

4.2 Une linéarisation améliorée

Dans le programme linéaire précédent, les contraintes (4.9) pèse lourdement sur la taille du problème, car il y a autant de contraintes que d'historiques joints terminaux. Pour alléger la résolution, il est possible de s'appuyer sur des considérations sur le nombre d'historiques terminaux. Nous sommes dans le cadre de politiques pures, un agent a donc $|O_i|^{T-1}$ historiques terminaux et le nombre d'historiques joints terminaux est $\prod_{i\in I}|O_i|^{T-1}$.

Dès lors, l'affirmation "*un historique terminal est dans le support de la politique d'un agent i ou il ne l'est pas*" peut être remplacée par "*le nombre d'historiques joints dans lequel se trouve un historique h_i d'un agent i vaut soit $\frac{\prod_{k\in I}|O_k|^{T-1}}{|O_i|^{T-1}}$ soit* 0". C'est dans cet esprit que nous proposons de remplacer la contrainte :

$$\sum_{i=1}^{n} x_i(j_i) - nz(j) \quad \geq \quad 0, \quad \forall j \in \mathcal{E} \tag{4.13}$$

par les contraintes :

$$\sum_{j'\in H^T_{-i}} z(h,j') \quad = \quad \frac{\prod_{k\in I}|O_k|^{T-1}}{|O_i|^{T-1}} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i. \tag{4.14}$$

On obtient ainsi un nouveau programme linéaire entier qui s'écrit :

Programme Linéaire Entier Amélioré

$$\text{maximiser} \qquad \sum_{j\in\mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.15}$$

sous les contraintes,

$$C_i x_i \quad = \quad c_i, \quad \forall i \in I \tag{4.16}$$

$$\sum_{j\in H^T} z(j) \quad = \quad \prod_{i\in I}|O_i|^{T-1} \tag{4.17}$$

$$\sum_{j'\in H^T_{-i}} z(h,j') \quad = \quad \frac{\prod_{k\in I}|O_k|^{T-1}}{|O_i|^{T-1}} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.18}$$

$$x_i(h) \quad \in \quad \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{4.19}$$

$$z(j) \quad \in \quad \{0,1\}, \quad \forall j \in \mathcal{E} \tag{4.20}$$

La section 4.3 du manuscrit anglais revient plus en détail sur cette transformation et montre que ce programme linéaire permet bien de résoudre le DEC-POMDP en s'appuyant sur le théorème suivant :

Théorème 4.2. *Une solution (x^*, z^*) du programme linéaire entier (4.15)-(4.20) permet de définir une politique joint optimale $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ pour le DEC-POMDP considéré.*

La démonstration de ce résultat, qui se trouve page 105 passe par la démonstration du lemme suivant :

Lemme 4.2. *Toute solution (x^*, z^*) du programme linéaire entier (4.15)-(4.20) satisfait la condtition suivante pour chaque historique joint terminal $j \in \mathcal{E}$,*

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I. \tag{4.21}$$

Ce lemme est démontré en page 104.

4.3 Vers des programmes linéaire mixtes entiers

En pratique, la résolution de programme linéaire mixte est coûteuse car les solveur sont presque obligés de tester les valeurs de chaque variable entière. Il est important de réduire le nombre de variables entières, en s'appuyant sur des considérations concernant la propagation des contraintes , de manière analogues à celles sur les contraintes de politiques (voir lemme 3.2.0.0.0).

Il est donc possible, pour les variables x représentant le poids des historiques, de n'imposer des variables entières que pour les historiques terminaux. Si on ne peut pas aller plus loin pour le premier programme linéare, il est possible de faire mieux pour le programme linéaire amélioré en relaxant aussi les contraintes sur les variables z.

On obtient alors les deux programmes linéaires mixtes entiers suivants. Nous avons montré en section 4.4.1 du manuscrit anglais que chacun de ces deux programmes permet de trouver une solution au DEC-POMDP.

Programme Linéaire Mixte Entier

$$\text{maximiser} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.22}$$

sous les contraintes,

$$C_i x_i = c_i, \quad \forall i \in I \tag{4.23}$$

$$\sum_{j \in H^T} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{4.24}$$

$$\sum_{i=1}^{n} x_i(j_i) - n z(j) \geq 0, \quad \forall j \in \mathcal{E} \tag{4.25}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{4.26}$$

$$x_i(h) \in \{0, 1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.27}$$

$$z(j) \in \{0, 1\}, \quad \forall j \in \mathcal{E} \tag{4.28}$$

Programme Linéaire Mixte Entier Amélioré

$$\text{maximiser} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.29}$$

sous les contraintes,

$$C_i x_i = c_i, \quad \forall i \in I \tag{4.30}$$

$$\sum_{j \in H^T} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{4.31}$$

$$\sum_{j' \in H^T_{-i}} z(h, j') = \frac{\prod_{k \in I} |O_k|^{T-1}}{|O_i|^{T-1}} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.32}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{4.33}$$

$$x_i(h) \in \{0, 1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.34}$$

$$z(j) \in [0, 1], \quad \forall j \in \mathcal{E} \tag{4.35}$$

4.4 Bilan

Ce chapitre a proposé deux programmes linéaires mixtes entier pour résoudre des DEC-POMDP à n agents. L'approch suivie à essentiellement consisté à linéariser la fonction objectif en s'appuyant sur le fait qu'il existait au moins une politique jointe optimale pure.

Dans le chapitre suivant, nous allons exploiter des propriétés propres à la théorie des jeux pour proposer de nouvelles formes, plus efficaces, de programmes mathématiques permettant de résoudre des DEC-POMDP.

Chapitre 5

Approche à base d'équilibre de Nash optimal

5.1 Définitions préliminaires

Plusieurs notions sont essentielles à la mise en œuvre des méthodes de ce chapitre. La plupart de ces notions sont inspirées de la littérature sur la théorie des jeux.

5.1.1 Meilleure réponse et équilibre de Nash

Une politique $p' \in X_i$ d'un agent i est une **meilleure réponse** à une politique jointe i-réduite $q \in X_{-i}$ si on a :

$$\mathcal{V}(\alpha, (p', q)) \quad \geq \quad \mathcal{V}(\alpha, (p'', q)), \quad \forall p'' \in X_i. \tag{5.1}$$

Une façon de qualifier un équilibre de Nash, qui est une notion essayant de définir un comportement collectif rationnel, est de dire que c'est une politique jointe où chaque politique individuelle est une meilleure réponse à toutes les autres politiques. Formellement, une politique jointe $p \in X$ est un **équilibre de Nash** si l'on a :

$$\mathcal{V}(\alpha, p) \quad \geq \quad \mathcal{V}(\alpha, (p', p_{-i})), \quad \forall i \in I, \forall p' \in X_i \tag{5.2}$$

Une politique jointe optimale est donc clairement un équilibre de Nash mais l'inverse n'est pas vrai car la valeur d'un équilibre de Nash peut être moins élevée que celle de la politique optimale. Par exemple, dans l'exemple ci-dessous où chacun des deux agents dipose de trois politiques $\{a, b, c\}$, la politique jointe (b,b) n'est pas un équilibre de Nash car `agent2` peut faire mieux en jouant c. Par contre, (a,a) et (c,c) sont deux équilibres de Nash mais seul (c,c) est "optimal".

		agent 2		
		a	b	c
agent 1	a	1	0	0
	b	0	2	3
	c	0	3	4

5.1.2 Regret d'un historique

La **valeur d'un ensemble d'information** $\iota \in \mathcal{I}_i$ d'un agent i pour une politique jointe i-réduite q, notée $\lambda_i^*(\iota, q)$, est définie par :

$$\lambda_i^*(\iota, q) = \max_{h \in \iota} \sum_{j' \in \mathcal{E}_{-i}} \overline{V}(\alpha, (h, j'))q(j') \tag{5.3}$$

et, si ι est un ensemble d'information non-terminal,

$$\lambda_i^*(\iota, q) = \max_{h \in \iota} \sum_{o \in O_i} \lambda_i^*(ho, q) \tag{5.4}$$

Quant au **regret d'un historique** h d'un agent i pour une politique jointe i-réduite q, noté $\mu_i(h, q)$, nous le définisson par :

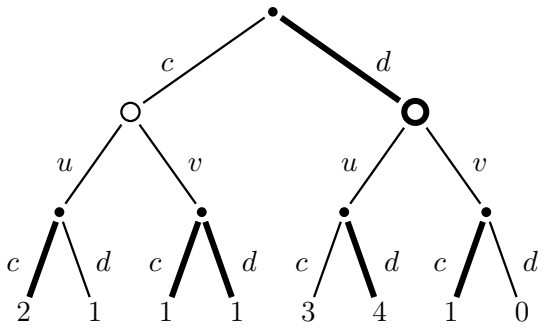
$$\mu_i(h, q) = \lambda_i^*(\iota(h), q) - \sum_{j' \in H_{-i}^T} \overline{V}(\alpha, (h, j'))q(j') \tag{5.5}$$

et, si h est un historique non-terminal,

$$\mu_i(h, q) = \lambda_i^*(\iota(h), q) - \sum_{o \in O_i} \lambda_i^*(ho, q) \tag{5.6}$$

Ces deux concepts sont indépendants de la politique de l'agent i et permettent de mesurer l'impact des décisions de l'agent par rapport à ses actions optimales.

L'exemple ci-dessous considère un agent dont l'ensemble d'action est $A_i = \{c, d\}$ et l'ensemble des obserations est $O_i = \{u, v\}$. Pour chaque historique h de taille $T = 2$, le nombre indiqué est la contribution $\sum_{j' \in \mathcal{E}_{-i}} \overline{V}(\alpha, (h, j'))q(j')$ de cet historique en supposant que la politique i-réduite associée est q.



La valeur de l'ensemble d'information cu est 2 car la valeur maximale qu'il peut y obtenir, en choisissant l'action b, est 2. La valeur de cv est 1, *etc.* La valeur de l'ensemble d'information $\varnothing$ est la plus grande valeur entre $(\lambda_i^*(cu, q) + \lambda_i^*(cv, q))$ et $(\lambda_i^*(du, q) + \lambda_i^*(dv, q))$, c'est-à-dire 5.

Les traits en gras indiquent les historiques qui sont de regret 0 et on peut en déduire une meilleure réponse qui serait alors $p(d) = 1$, $p(dud) = 1$, $p(dvc) = 1$ et $p(h) = 0$ pour tout autre historique.

5.1.3 Des contraintes complémentaires comme conditions nécessaires

Le but est de trouver une formulation sous forme d'un programme linéaire de la solution d'un DEC-POMDP. Les détails de cette dérivation sont donnés section 5.4 du manuscrit anglais), page 121.

Puisque la solution optimale est le "meilleur" équilibre de Nash, nous allons d'abord chercher à caractériser un tel équilibre sans avoir à chercher un maximum.

Commençons tout d'abord par exprimer le fait qu'une solution est un équilibre de Nash, donc que pour un agent i, sa politique x_i est une meilleure réponse à la politique q des autres agents. Nous avons donc, pour *un agent* i, le programme mathématique suivant :

$$\text{maximiser} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) x_i(j_i) q(j'_{-i}) \tag{5.7}$$

sous les contraintes,

$$\sum_{a \in A_i} x_i(a) = 1 \tag{5.8}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{5.9}$$

$$x_i(h) \geq 0, \quad \forall h \in \mathcal{H}_i. \tag{5.10}$$

Dans ce programme, la politique de l'agent i est codée par les variables $x_i(h)$ qui représentent les poids des différents historiques. Ce problème peut s'exprimer de manière duale par :

$$\text{minimiser} \quad y_i(\varnothing) \tag{5.11}$$

sous les contraintes,

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) \geq 0, \quad \forall h \in \mathcal{N}_i \tag{5.12}$$

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) q(j'_{-i}) \geq 0, \quad \forall h \in \mathcal{E}_i. \tag{5.13}$$

Dans le problème dual, chaque variable $y_i(\iota)$ représente la valeur de l'ensemble d'informaion ι. Ainsi, les membres de gauche des contraintes (5.12) et (5.13) sont assimilées au regret des historiques qui leur sont associés.

Le théorème de la dualité en programmation linéaire [Lue84] nous indique que les valeurs des fonctions objectifs des solutions des deux programmes sont égales. De part cette égalité et en exploitant la positivité de certaines contraintes, si les solutions sont x^* et y^* nous avons :

$$x_i^*(h)\big\{y_i^*(\iota(h)) - \sum_{o \in O_i} y_i^*(ho)\big\} = 0, \quad \forall h \in \mathcal{N}_i \tag{5.14}$$

$$x_i^*(h)\big\{y_i^*(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) q(j'_{-i})\big\} = 0, \quad \forall h \in \mathcal{E}_i. \tag{5.15}$$

Il nous reste à exprimer une politique jointe i-réduite $q(j'_{-i})$ comme étant le produit des poids des politiques individuelles pour les historiques support, c'est-à-dire que :

$$q(j'_{-i}) = \prod_{k \in I \setminus \{i\}} x_k(j'_k). \tag{5.16}$$

En intégrant cette définition, en utilisant la propriété des solutions des progrmme duals comme une conditions nécessaire à l'obtention d'une politique jointe optimale, on obtient des contraintes qui permettent de caractériser une politique jointe qui est un équilibre de Nash, à savoir :

$$C_i x_i = c_i, \quad \forall i \in I \tag{5.17}$$

$$x_i(h)\big\{y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho)\big\} = 0, \quad \forall h \in \mathcal{N}_i \tag{5.18}$$

$$x_i(h)\big\{y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} x_k(j'_k)\big\} = 0, \quad \forall h \in \mathcal{E}_i \tag{5.19}$$

$$x_i(h) \geq 0, \quad \forall h \in \mathcal{H}_i. \tag{5.20}$$

Cependant, le théorème de la dualité en programmation linéaire nous indique que la valeur de la politique est aussi *la valeur de l'ensemble d'information nul.* on a donc :

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{k \in I} x_k(j_k) = y_i(\varnothing) \tag{5.21}$$

. Il est alors possible de résoudre un DEC-POMDP en cherchant la solution du programme avec contraintes complémentaires ci-dessous qui recherche le "meilleur équilibre de Nash joint".

Programme avec Contraintes Complémentaires

$$\text{maximiser} \quad y_1(\varnothing) \tag{5.22}$$

avec, pour chaque agent $i \in I$, les contraintes suivantes :

$$C_i x_i = c_i, \quad \forall i \in I \tag{5.23}$$

$$x_i(h)\big\{y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho)\big\} = 0, \quad \forall h \in \mathcal{N}_i \tag{5.24}$$

$$x_i(h)\big\{y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} x_k(j'_k)\big\} = 0, \quad \forall h \in \mathcal{E}_i \tag{5.25}$$

$$x_i(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{5.26}$$

Dans ce programme linéaire, les variables x_i représentent encore une fois les poids des historiques, ce qui permet de trouver la politique optimale dont la valeur est exprimée par la valeur de l'ensemble d'information $\varnothing$, c'est-à-dire la valeur de $y_1(\varnothing)$. Il reste malheureusement que l'aspect non-linéaire des contraintes complémentaires (5.24) et (5.25) rend difficile la recherche de la solution optimale. Nous allons voir comment linéarises ces contraintes.

5.2 Séparer les contraintes linéaires

Pour linéariser les contraintes complémentaires précédentes, une méthode générale s'appuie sur une connaissance des bornes supérieures et inférieures des valeurs que peuvent prendre les variables.

En effet, imaginons la contrainte complémentaire $ab = 0$. Si la borne inférieur de a et b est 0, si les bornes supérieurs de a et b sont respectivement u_a et u_b, il est possible d'utiliser une variables entière c à valeur dans $\{0; 1\}$ pour écrire que la contrainte est équivalente à :

$$\begin{aligned} a &\leq u_a c && (5.27)\\ b &\leq u_b(1-c) && (5.28) \end{aligned}$$

Il nous reste maintenant à trouver des bornes supérieure et inférieures pour les contraintes (5.24) et (5.25) précédentes, c'est-à-dire des bornes pour les regrets des historiques terminaux et non-terminaux. Le calcul de ces bornes est détaillé dans la section 5.5 du manuscrit anglais, page 123. D'une part, nous y vérifions que les bornes inférieurs sont bien 0 et d'autre part, nous y démontrons les deux propiétés suivantes.

Borne pour historique terminal. *$\mathcal{U}_i^T(h)$ définit comme suit est une borne supérieure du regret d'un historique terminal h de l'agent i.*

$$\mathcal{U}_i^T(h) = \prod_{k\in I\setminus\{i\}} |O_k|^{T-1}\{\max_{h'\in\iota(h)}\max_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h',j')) - \min_{j''\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h,j''))\} \quad (5.29)$$

Borne pour historique non-terminal. *$\mathcal{U}_i(h)$ définit comme suit est une borne supérieure du regret de l'historique non-terminal t de longueur t pour l'agent i.*

$$\mathcal{U}_i(h) = L_i\{\max_{h'\in\mathcal{E}_{(\iota(h))}}\max_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h',j')) - \min_{g\in\mathcal{E}_i(h)}\min_{j''\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(g,j''))\} \quad (5.30)$$

où,

$$L_i = |O_i|^{T-t}\prod_{k\in I\setminus\{i\}} |O_k|^{T-1} \quad (5.31)$$

Ainsi la contrainte

$$x_i(h)\big\{y_i(\iota(h)) - \sum_{o\in O_i} y_i(ho)\big\} = 0 \quad (5.32)$$

est séparée en une paire de contraintes linéaires en utilisant des variables $b_i(h)$ à valeur dans $\{0; 1\}$ avec

$$\begin{aligned} x_i(h) &\leq 1 - b_i(h) && (5.33)\\ y_i(\iota(h)) - \sum_{o\in O_i} y_i(ho) &\leq \mathcal{U}_i(h)b_i(h) && (5.34)\\ b_i(h) &\in \{0,1\}. && (5.35) \end{aligned}$$

De manière similaire, la contrainte

$$x_i(h)\big\{y_i(\iota(h)) - \sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h,j')) \prod_{k\in I\setminus\{i\}} x_k(j'_k)\big\} \;=\; 0 \tag{5.36}$$

s'exprime comme

$$x_i(h) \;\leq\; 1 - b_i(h) \tag{5.37}$$

$$y_i(\iota(h)) - \sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h,j')) \prod_{k\in I\setminus\{i\}} x_k(j'_k) \;\leq\; \mathcal{U}_i^T(h) b_i(h) \tag{5.38}$$

$$b_i(h) \;\in\; \{0,1\}. \tag{5.39}$$

5.3 Un programme mixte entier pour 2 agents

Quand on ne considère que deux agents, le programme mathématique avec des contraintes complémentaires vu précédemment se réécrit plus simplement en séparant la seule paire de contrainte complémentaire. Cela donne le programme linéaire mixte entier suivant à résoudre.

Programme Linéaire Mixte Entier pour 2 agents

$$\text{maximiser} \quad y_1(\varnothing) \tag{5.40}$$

avec, pour chaque agent $i \in \{1,2\}$, les contraintes suivantes :

$$C_i x_i \;=\; c_i \tag{5.41}$$

$$x_i(h) \;\leq\; 1 - b_i(h), \quad \forall h \in \mathcal{H}_i \tag{5.42}$$

$$y_i(\iota(h)) - \sum_{o\in O_i} y_i(ho) \;\leq\; \mathcal{U}_i(h) b_i(h), \quad \forall h \in \mathcal{N}_i \tag{5.43}$$

$$y_i(\iota(h)) - \sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h,j')) x_{-i}(j'_{-i}) \;\leq\; \mathcal{U}_i^T(h) b_i(h), \quad \forall h \in \mathcal{E}_i \tag{5.44}$$

$$x_i(h) \;\geq\; 0, \quad \forall h \in \mathcal{H}_i \tag{5.45}$$

$$b_i(h) \;\in\; \{0,1\}, \quad \forall h \in \mathcal{H}_i \tag{5.46}$$

avec $\mathcal{U}_i$ et $\mathcal{U}_i^T$ définis précédemment (équations 5.30 et 5.29).

Et, en section 5.6 du manuscrit anglais, page 126, nous prouvons le théorème suivant qui assure l'optimalité de la solution.

Théorème 5.2. *Une solution (x^*, y^*, b^*) du programme linéaire mixte entier (5.40)-(5.46), définit une politique jointe optimale $x^* = (x_1^*, x_2^*)$.*

Ce théorème est prouvé page 126.

5.4 Un programme mixte entier pour 3 agents et plus

Quand on considère plus de deux agents, chaque contrainte complémentaire ne peut être entièrement séparée en contraintes linéaires car il reste toujours un terme non-linéaire $\sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h,j')) \prod_{k\in I\setminus\{i\}} x_k(j_k)$. Il faut donc remplacer aussi ces termes non-linéaire par des contraintes linéaires.

La façon de procéder, qui est détaillée en section 5.7 du manuscrit anglais, est proche de la méthode employée précédemment pour linéariser de manière efficace le programme non-linéaire générique d'un DEC-POMDP sous forme séquentielle (voir section 4.2). On restreint l'espace de recherche aux politiques pures, on utilise les contraintes sur le nombre d'historiques terminaux pour remplacer chaque terme

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E} H_{-i}} \overline{V}(\alpha, (h, j'))x_{-i}(j') \tag{5.47}$$

par des termes de la forme

$$y_i(\iota(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \overline{V}(\alpha, (h, j_{-i}))z(j). \tag{5.48}$$

Dès lors, le programme linéaire mixte entier suivant permet de trouver une solution au DEC-POMDP considéré.

Programme Linéaire Mixte Entier pour $n \geq 2$ agents

$$\text{maximiser} \quad y_1(\varnothing)$$

sous les contraintes, pour chaque agent $i \in I$,

$$\begin{aligned}
C_i x_i &= c_i \\
x_i(h) &\leq 1 - b_i(h), \quad \forall h \in \mathcal{H}_i \\
y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) &\leq \mathcal{U}_i(h) b_i(h), \quad \forall h \in \mathcal{N}_i \\
y_i(\iota(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, (h, j_{-i}))z(j) &\leq \mathcal{U}_i^T(h) b_i(h), \quad \forall h \in \mathcal{E} i \\
\sum_{j' \in \mathcal{E}_{-i}} z(h, j') &= \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall h \in \mathcal{E}_i \\
\sum_{j \in \mathcal{E}} z(j) &= \prod_{i \in I} |O_i|^{T-1} \\
x_i(h) &\geq 0, \quad \forall h \in \mathcal{N}_i \\
x_i(h) &\in \{0, 1\}, \quad \forall h \in \mathcal{E}_i \\
b_i(h) &\in \{0, 1\}, \quad \forall h \in \mathcal{H}_i \\
z(j) &\in [0, 1], \quad \forall j \in \mathcal{E}
\end{aligned}$$

avec $\mathcal{U}_i$ et $\mathcal{U}_i^T$ défini précédemment (équations 5.30 et 5.29).

Cette propriété est garantie par le théorème ci-dessous.

Théorème 5.3. *Une solution (x^*, y^*, b^*, z^*) du programme linéaire mixte entier (5.49)-(5.59) définit une politique jointe optimale $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$.*

Ce théorème est démontré à la page 128.

5.5 Bilan

Dans ce chapitres nous avons présentés des programmes mathématiques pour résoudre des DEC-POMDP avec 2 agents ou, pour la deuxième versions, 3 agents ou plus. Le nombre de variables pour le premier programme linéaire mixte entier est exponentiel en l'horizon T alors que ceux du chapitre précédent l'était en T *et* n, le nombre d'agent. Cette version est beaucoup plus efficace, comme montré par nos expérimentations. Elle permet de plus de résoudre les *jeux stochastiques partiellement obervables*, ainsi que nous l'indiquons en annexe.

Le deuxième programme, qui peut gérer plus d'agents, est moins efficace et se restreint aux politiques pures. Il peut aussi être utilisé pour résoudre des jeux stochastiques partiellement obervables mais en ne recherchant que des politiques pures. Ce qui est un apport intéressant quand on considère qu'il n'existe qu'un seul autre algorithme pour ce genre de problème [GW01].

Chapitre 6

Heuristiques et Programmation Dynamique

Ce chapitre, qui est le pendant du Chapitre 6 de la version anglaise du manuscrit que l'on trouvera à la page 133, propose des méthodes heuristiques pour diminuer le temps de résolution des programmes linéaires proposé dans cette thèse.

6.1 Historiques localement superflus

Un historique **localement superflu** est un historique dont on peut prouver à l'avance qu'il n'est pas requis pour exprimer une politique jointe optimale en partant de l'état α. On peut toujours remplacer un historique localement superflu par un de ses co-historique. Un **co-historique** d'un historique h pour un agent i est un historique qui est en tout point similaire à h sauf pour sa dernière action. On note $C(h)$ l'ensemble des co-historiques de h.

On peut alors définir plus formellement un historique localement superflu. Un historique $h \in \mathcal{H}_i^t$ de longueur t d'un agent i est **localement superflu** si pour toute distribution de probabilité γ sur l'ensembleover $\mathcal{H}_{-i}^t$ des historiques joints i-réduits de longueur t, il existe un co-historique $h' \in C(h)$ tel que,

$$\sum_{j' \in \mathcal{H}_{-i}^t} \gamma(j') \big\{ \mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j')) \big\} \quad \geq \quad 0 \tag{6.1}$$

où $\gamma(j')$ denote la probabilité de j' selon γ.

Une deuxième définition équivalent dit que un historique $h \in \mathcal{H}_i^t$ de longueur t d'un agent i est **localement superflu** s'il existe une distribution de probabilité ω sur l'ensemble $C(h)$ des co-historiques de h telle que pour chaque historique joint i-réduit j' de taille t, on a :

$$\sum_{h' \in C(h)} \omega(h') \mathcal{R}(\alpha, (h', j')) \geq \mathcal{R}(\alpha, (h, j')) \tag{6.2}$$

où $\omega(h')$ est la probabilité du co-historique h' selon ω.

Ainsi que le montre le théorème suivant, dans une politique jointe optimale on peut toujours remplacer un historiqur localement superflu par un de ses co-historique sans affecter le caractère optimal de cette politique.

Théorème 6.1. *Pour chaque politique jointe optimale p' d'horizon T telle qu'il existe un agent $i \in I$ ayant un historique terminal h qui est localement superflu en α et dont $p'_i(h) > 0$, il exite une autre politique jointe p d'horizon T optimale en α qui est identique à p' mais où $p_i(h) = 0$.*

Ce théorème est démontré à la page 134. Ce théorème implique que deux cas peuvent se produire en fait :

- cas (i) : tous les co-historiques de h sont aussi localement superflus.
- cas (ii) : il y a au moins un co-historique de h qui n'est pas localement superflu.

6.2 identifer et éliminer les historiques localement superflus

Il y a deux moyens complémentaires de détecter (et donc d'éliminer) les historiques qui sont localement superflus. Ces deux techniques sont détaillée dans les section 6.3 du manuscrit anglais, pages 135.

Pour savoir si h est localement superflu on procède en deux temps. Dans un premier temps, l'idée est de vérifier que la probabilité *a priori* de tous les historiques joints où apparaît l'historique h n'est pas nulle. On vérifie donc si :

$$\Psi(\alpha, (h, j')) \quad = \quad 0, \quad \forall j' \in \mathcal{H}^t_{-i}. \tag{6.3}$$

Ce test est une condition suffisante mais pas nécessaire, il ne détecte pas tous les historiques localement superflus. Un test plus coûteux mais plus efficace passe par l'utilisation d'un programme linéaire qui s'appuie sur la définition d'un historique localement superflu. Comme démontré à la page 136 du mansucrit anglais, si la fonction objectif ϵ^* solution du programme suivant est positive, alors h est localement superflu.

$$\text{Minimiser} \quad \epsilon \tag{6.4}$$

, Avec les contraintes

$$\sum_{j' \in \mathcal{H}^t_{-i}} y(j') \big\{ \mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j')) \big\} \quad \leq \quad \epsilon, \quad \forall h' \in C(h) \tag{6.5}$$

$$\sum_{j' \in \mathcal{H}^t_{-i}} y(j') \quad = \quad 1 \tag{6.6}$$

$$y(j') \quad \geq \quad 0, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{6.7}$$

Enfin, il est important de noter que si tous les descendants *hoa* de h sont lolement superflu, alors h est lui-aussi localement superflu. La dernière phase du processus d'élimination consiste donc à eliminer récursivement tous les historiques dont *tous* les descendant sont localement superflus.

6.3 Historiques globalement superflus

La notion d'historique globalement superflu est très similaire à celle d'historique localement superflu. Elle est plus contraignante et plus coûteuse à vérifer cependant.

Une politique globalement superflue n'est pas nécessaire pour exprimer une politique optimale jointe *quel que soit l'état initial* du DEC-POMDP.

La définition formelle de cette propriété s'appuie sur le fait que la valeur d'un historique joint en chaque état de $\Delta(S)$ est une combinaison linéaire des valeurs des somments de $\Delta(S)$, c'est-à-dire les états de S. Ainsi, un historique $h \in \mathcal{H}_i^t$ de longueur t d'un agent i est dit **globalement superflu** si pour chaque distribution de probabilité γ sur l'ensemble $\mathcal{H}_{-i}^t \times S$ il existe un co-historique h' de $C(h)$ tel que :

$$\sum_{s \in S} \sum_{j' \in \mathcal{H}_{-i}^t} \gamma(j', s)\big\{\mathcal{R}(s, (h', j')) - \mathcal{R}(s, (h, j'))\big\} \quad \geq \quad 0 \tag{6.8}$$

où $\gamma(j', s)$ est la probabilité de la paire (j', s) selon γ.

Il y a aussi une définition alternative qui est la suivante. Un historique $h \in \mathcal{H}_i^t$ de longueur t d'un agent i est dit **globalement superflu** si il existe une distribution de probabilité ω sur l'ensemble des co-histoires de h telle que pour chaque historique jointe i-réduite j' de longueur t et pour chaque état $s \in S$, on a :

$$\sum_{h' \in C(h)} \omega(h')\mathcal{R}(s, (h', j')) \geq \mathcal{R}(s, (h, j')) \tag{6.9}$$

où $\omega(h')$ est la probailité du co-historique h' selon ω.

Comme précédemment, un théorème nous permet de montrer que l'on peut se passer des historiques globalement superflus.

Théorème 6.2. *Pour tout état $\beta \in \Delta(S)$, pour toute politique jointe p' d'horizon t optimale en β telle qu'il existe un agent $i \in I$ et un historique h de longueur t de cet agent i qui est globalement superflu avec $p'_i(h) > 0$, il existe une autre politique jointe p d'horizon t qui est optimale en α et qui est identique à p' sauf pour $p_i(h) = 0$.*

Ce théorème est démontré à la page 139. La encore, deux cas peuvent se produire si h est globalement superflu.

- cas (i) : tous les co-historiques de h sont globalement superflus.
- cas (ii) : il existe au moins un co-historique qui n'est pas globalement superflu.

6.4 Identifier et éliminer les historiques globalement superflus

On suit une démarche similaire à celle des histoqiques localement superflus, mais plus compliquée et plus coûteuse. D'un part, si la probabilité de la séquence d'observatio jointe est nulle, nous avons que h est globalement superflu. Ce qui s'écrit :

$$\Psi(\alpha, (h, j')) \quad = \quad 0, \quad \forall j' \in \mathcal{H}_{-i}^t \tag{6.10}$$

Comme ce test ne suffit pas à détecter tous les historiques globalement superflus, nous avons ensuite recours à un programme linéaire qui s'appuie sur la définition de ces historique. Un historique h est éliminé si la valeur de la fonction objectif ϵ du programme linéaire suivant est positive.

$$\text{Minimiser} \quad \epsilon \tag{6.11}$$

Sous les contraintes,

$$\sum_{s\in S}\sum_{j'\in\mathcal{H}^t_{-i}} y(s,j')\big\{\mathcal{R}(s,(h',j')) - \mathcal{R}(s,(h,j'))\big\} \;\leq\; \epsilon, \quad \forall h' \in C(h) \tag{6.12}$$

$$\sum_{j'\in\mathcal{H}^t_{-i}} y(j') \;=\; 1 \tag{6.13}$$

$$y(j') \;\geq\; 0, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{6.14}$$

Comme le détaille la section 6.5 du manuscrit anglais, page 140, la procédure de recherche et d'élimination des historiques globalement superflus est très proche de l'algorithme de Programmation Dynamique de [HBZ04].

6.5 Modification des Programmes Linéaires Mixtes Entiers

Après avoir éliminé des historiques, nous ne travaillons plus avec les ensembles $\mathcal{H}_i$ des historiques de taille inférieure ou égale à T mais avec un de ses sous ensemble qui est celui des historiques de taille inférieure ou égale à T *qui ne sont pas localement superflus.* pour l'agent i, cet ensembles est noté $\tilde{H}_i$ et l'ensemble des historiques non-superflus terminaux sera noté $\tilde{H}_i^T$. De manière similaire, l'ensemble des historiques de taille inférieur ou égals à T qui ne sont pas globalement superflus est est noté $\overline{H}_i$ et $\overline{H}_i^T$ quand on ne parle que des historiques terminaux.

Dans les programmes linéaires proposés aux chapitres précédents, il faut modifier certaines contraintes pour prendre en compte ces ensembles d'historiques dont la taille est moindre. En particulier, nous avons souvent utilisé des arguments de dénombrement sur le nombre d'historiques qui se traduisaient par des contraintes du type suivant dans les programmes linéaires :

$$\sum_{j\in\mathcal{E}} z(j) \;=\; \prod_{i\in I} |O_i|^{T-1}$$

en s'appuyant sur le fait qu'il y a $|O_i|^{T-1}$ historiques terminaux pour l'agent i. Cette contrainte doit être relaxée un peu pour prendre en compte que, comme nous allons travailler avec des sous-ensembles d'historiques, il y aura moins de variables libres dans les programmes mathématiques. La contrainte précédente deviend donc :

$$\sum_{j\in H^T} z(j) \;\leq\; \prod_{i\in I} |O_i|^{T-1}. \tag{6.15}$$

Comme le détaille la section 6.6 du manuscrit anglais, d'autres contraintes du même type doivent aussi être modifiées dans les programmes linéaires **MILP1** à **MILP5**. Il faut parfois modifier aussi la fonction objectif de ces programmes. Par exemple, pour les programmes **MILP1** et **MILP2**, la fonction objectif qui était :

$$\text{Maximiser} \qquad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j)$$

devient :

$$\text{Maximiser} \qquad \sum_{j \in \overline{H}} \sum_{s \in S} \alpha(s) \mathcal{R}(s, j) z(j) \tag{6.16}$$

6.6 Coupures

Les heuristiques qui viennent d'être présentées ont pour but de réduire la taille des programmes linéaire, ce qui permet de diminuer le temps de résolution de ces programmes. Une autre façon de faire est d'introduire des plans de coupure, ou coupures, dans la résolution. Une coupure est contrainte qui réduit l'espace de recherche en écartant une zone où on peut prouver que la solution optimale ne se trouve pas [Dan60]. Dans le cas des DEC-POMDP, deux coupures peuvent être définie.

Nous proposons d'abord de couper l'espace des solution en s'appuyant sur une **borne supérieure** de la valeur du DEC-POMDP. En effet, la valeur d'un DEC-POMDP est en effet majorée par la valeur du POMDP "sous-jacents", c'est-à-dire le même problème mais comme s'il était résolu de manière centralisée. Il est aussi possible de borner la valeur du DEC-POMDP par celle du MDP sous jacent, mais cette borne est moins précise.

La section 6.7.1 du manuscrit anglais montre que l'on peut résoudre le POMDP associé au DEC-POMDP avec un programme linéaire plus simple et qui ne consomme qu'un 1% du temps de résolution du DEC-POMDP.

Il est aussi possible, mais moins immédiat, de proposer une borne inférieure à la valeur du DEC-POMDP. Si les récompenses étaient toujours positives, la valeur de la politique optimale d'horizon T serait supérieur à la valeur de la politique optimale d'horizon $T-1$. Dans le cas le plus général, où les récompenses peuvent être négatives, on peut néanmoins calculer une borne inférieure en ajoutant à la politique optimale d'horizon $T-1$ le *minimum* de la récompense que l'on peut obtenir en une action. Plus formellement, on a :

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \quad \geq \quad \mathcal{V}(\alpha, p^*) + \min_{a \in A} \min_{s \in S} R(s, a) \tag{6.17}$$

où p^* est la politique optimale d'horizon $T-1$.

Cette borne inférieur peut se calculer avec n'importe lequel des programmes linéaires introduits dans cette thèse, en utilisant de nouveaux des bornes inférieures, d'où une récursion qui n'est pas sans rappeler l'algorithme MAA* vu au Chapitre 2.

6.7 Bilan

Dans ce chapitre, nous avons proposé plusieurs solutions “pratiques” pour diminuer le temps de résolution des différents programmes linéaires de cette thèse. Dans un premier temps, nous avons proposé d'éliminer les historiques “superflus” qui ne seront pas dans le suppert de la politique jointe optimale. Dans un deuxième temps, nous avons proposé des plans de coupure dont le but est de réduire l'espace de recherche des programmes linéaires. La chapitre suivant, qui présente des résultats expérimentaux, montre l'efficacité de ces techniques.

Chapitre 7

Expériences

7.1 Les différents programmes mathématiques

Dans les chapitres précédents, nous avons proposés plusieurs programmes mathématiques pour résoudre des DEC-POMDP. Ces programmes ont des propriétés différentes qui sont résumées ici, n représentant un entier supérieur ou égal à 2.

Chapitre	Programme	Type	Label	Commentaire
3	(3.20)-(3.22)	Programme Non-Linéaire	**NLP1**	Localement Optimal, n agents
4	(4.58)-(4.65)	0-1 MILP	**MILP1**	Optimal, n agents
4	(4.68)-(4.75)	0-1 MILP	**MILP2**	Optimal, n agents
5	(5.68)-(5.79)	0-1 MILP	**MILP3**	Optimal, 2 agents
5	(5.90)-(5.104)	0-1 MILP	**MILP4**	Optimal, n agents
5	(5.110)-(5.124)	0-1 MILP	**MILP5**	Optimal, n agents

- Programme **NLP1**, (3.18)-(3.20), Chapitre 3, page 22 : l'objectif est une fonction non-linéaire et on ne peut donc garantir que l'on trouvera une solution globalement optimale.
- Programme , (5.40)-(5.46), Chapitre 5, page 34 : MILP de petit taille, donc efficace, mais applicable à 2 agents seulement.
- Programme (4.29)-(4.35), Chapitre 4, page 27 : MILP amélioré à partir du MILP précédent, reste d'une taille trop grande pour être efficace, *il ne sera pas testé*.
- Programme **M2**,(5.49)-(5.59), Chapitre 5, page 35 : MILP efficace utilisable pour plus de 2 agents. Cependant, pour des problèmes à 3 agents, sa consommation mémoire est telle qu'il ne peut s'appliquer qu'à des problèmes minuscule.

Le tableau suivant récapitule la taille de ces problèmes mathématiques, en fonction de T l'horizon du problème et n le nombre d'agents. La taille d'un programme s'exprime essentiellement en fonction du nombre de variable et de contraintes, mais le nombre de variables entièr est un facteur très important pour les programmes mixtes entier. Une description plus détaillée de la taille de ces programmes se trouve à la section 7.2 du manuscrit anglais, page 149.

Program	# Variables	# Constraints	# 0-1 Variables
NLP1	510	255	0
MILP1	$>$ 2 million	$>$ 2 million	384
MILP2	$>$ 2 million	639	384
MILP4	$>$ 2 million	2169	1020
MILP5	50937	2553	1788

Taille des programmes pour 3 agents, 2 actions, 2 observation, horizon 4

Program	# Variables	# Constraints	# 0-1 Variables
NLP1	Exp. in T	Exp. in T	0
MILP1	Exp. in T and n	Exp. in T and n	Exp. in T
MILP2	Exp. in T and n	Exp. in T	Exp. in T
MILP3	Exp. in T	Exp. in T	Exp. in T
MILP4	Exp. in T and n	Exp. in T	Exp. in T
MILP5	Exp. in T and n	Exp. in T	Exp. in T

Pour se faire une meilleur idée de la taille de ces problème, la table suivante donne les valeurs concrète de ces taille pour un DEC-POMDP où chacun des 3 agents a 2 actions et 2 observation et l'horizon est de 4.

7.2 Expérimentations

Nous avons testé les programmes **MILP2** et **MILP3** sur deux problèmes de la littérature et un nouveau problème plus artificiel mais légèrement plus conséquent.

- Le premier problème est celui du "Tigre multi-agent" (MA-Tiger) [NTY+03]. Ce problème est composé de 2 états, 3 actions et 2 observations par agent. Il est décrit en section 7.4 du manuscrit anglais, page 154.
- Le deuxième problème, est celui de l'allocation de canal pour des communications broadcast (MABC) [HBZ04]. Ce problème est composé de 4 états, 2 actions et 2 observations par agent. Il est décrit en section 7.5 du manuscrit anglais, page 157.
- Le troisème problème est composé de DEC-POMDP de 2 ou 3 agents générés aléatoirement. Ces problèmes restent petits (moins de 10 états) avec 2 actions et 2 observations par agents. L'horizon est de 4 pour les problèmes à 2 agents et de 3 pour les problèmes à 3 agents. Ces problèmes sont décrits en section 7.6 du manuscrit anglais, page 159.

Les deux premiers problèmes, bien que largement utilisés dans la littérature, ne peuvent pas vraiment être considérés comme des "benchmark". Nous reviendrons dans les sections suivantes sur les raisons qui nous poussent à faire cette remarque. C'est pourquoi nous avons voulu tester nos programme dans un cadre plus artificiel, plus complexe et moins "complaisant".

Nous avons limité nos tests au deux programmes **MILP2** et **MILP3** pour plusieurs raisons. En fait, pour des raisons de place mémoire, nous avons essentiellement testé nos programmes sur des problèmes avec deux agents. D'ailleurs, les tests que nous avons réalisés sur des problèmes à 3 agents confirment que, pour des horizons supérieurs à 3, les programmes sont trop gourmands en place mémoire pour être résolus..Nous étant limité à 2 agents, il paraissait logique de tester ce qui apparaissait comme la meilleure solution, c'est-à-dire **MILP3**. Dès lors, **MILP2** étant le meilleur programme obtenu par des considérations combinatoriale, nous avons voulu le comparer au programme précédent.

Fonction Réc.	Programme	Horizon T	Temps (Secs)	Horizon T	Temps (Secs)
A	**MILP2**	3	3.7	4	*
	MILP2-Low	3	4.9	4	72
	MILP2-Up	3	3.5	4	*
	MILP2-LOC	3	6.4	4	*
	MILP2-LOC-Low	3	7.6	4	175
	MILP2-LOC-Up	3	6.2	4	*
B	**MILP2**	3	0.95	4	*
	MILP2-Low	3	1.0	4	43
	MILP2-Up	3	1.6	4	*
	MILP2-LOC	3	3.6	4	*
	MILP2-LOC-Low	3	3.7	4	146
	MILP2-LOC-Up	3	4.3	4	*

TAB. 7.1 – Temps d'exécution de **MILP2** sur le problème du Tigre Multi-Agents.

Fonction Réc.	Programme	Horizon T	Temps (Secs)	Horizon T	Temps (Secs)
A	**MILP3**	3	11.16	4	*
B	**MILP3**	3	12.33	4	*

TAB. 7.2 – Temps d'exécution de **MILP3** sur le problème du Tigre Multi-Agents.

Ces deux programmes mathématiques ont été implanté en JAVA en utilisant le solveur `ILOG-CPLEX 10` pour résoudre les divers programmes mathématiques mixtes entiers à valeur dans 0 ou 1. L'ordinateur utilisé était un Pentium 4 Intel à 3,4 GHz et 2 Go de RAM.

Ces programmes ont été testé dans leur version originale telle que présentée dans les chapitres 4 ou 5 et en utilisant une ou plusieurs heuristiques parmi celles présentées au chapitre 6 et rappelées ci-dessous. Soit **M** l'un des deux programmes **MILP2** ou **MILP3**.

- **M** où l'on a enlevé les historiques localement superflus est noté **M**-LOC.
- **M** où l'on a enlevé les historiques globalement superflus est noté **M**-GLOB.
- **M** avec l'utilisation d'une borne supérieure de coupure est noté **M**-Up.
- **M** avec l'utilisation d'une borne inférieure de coupure est noté **M**-Low.

7.3 Problème du Tigre Multi-Agents

Nous avons testé nos programmes sur les deux versions du problème du Tigre Multi-Agents que l'on trouve dans la littérature, la fonction récompense étant l'unique différence entre les deux versions de ce problème.

Les tables 7.1 et 7.2 listent les différentes expériences réalisés sur ce problème. Nous avons testé plusieurs heuristiques pour des horizons de 3 ou 4 et pour les différentes fonctions de récompense. Les temps indiqué comprennet le temps de pré-traitement nécessaire pour les différentes heuristiques et si '*' indique un "time-out", ce dernier s'est toujours produit *après* le pré-traitement des heuristiques.

Fonction Récomp.	Algorithme	Type	Horizon	Temps	Horizon	Temps
A	MAA*	E	3	4 s	4	> 1 month
	Recursive MAA*	E	3	4 s	4	2.5 h
	MILP2	E	3	3.5 s	4	72 s
	Exhaustive-JESP	N	3	317 s	4	*
	DP-JESP	N	3	0	4	0.02 s
	MBDP	A	3	0.19 s	4	0.46 s
B	MAA*	E	3	1s	4	25 h
	Recursive MAA*	E	3	1s	4	25 h
	MILP2	E	3	0.95 s	4	43 s

TAB. 7.3 – Temps d'exécution des algorithmes existants sur le problème du Tigre Multi-agents.

Nous avons aussi comparé nos algorithmes avec les algorithmes de la littérature, qu'ils soient exacts ou non. Ainsi, dans la table 7.3, E indique un algotithme exact, N un algorithme localement exact (équilibre de Nash) et A un algorithme approché.

Il est intéressant de noter que même les algorithmes approchés trouvent une solution optimale pour ce problème. C'est ce qui nous fait dire que ce problème n'est pas forcément un benchmark intéressant car sa solution optimale n'est sans doute pas compliquée à trouver. On voit aussi que les heuristiques ne sont pas très utiles pour **MILP2** dont elles ralentissent l'exécution. Une analyse plus poussée nous a montré qu'il n'existait en fait aucun historique localement superflu pour ces problèmes. Enfin, **MILP3**, qui semblait plus prometteur car de plus petite taille, est en fait moins performant que **MILP2**. la raison principale semble être qu'il contient plus de variables qui doivent prendre des valeurs réelles, ce qui A FINIR.

7.4 Allocation de canal de communication

Le problème de l'allocation d'un canal de communication pour envoyer des messages en broadcast, que nous avons introduit dans le premier chapitre de ce manuscrit (section reffr :MABC), doit être légèrement modifié pour être modélisé comme un DEC-POMDP [HBZ04]. Comme expliqué dans la section 7.5 du manuscrit anglais, page 157, cette modélisation conduit à un DEC-POMDP à 4 états et 2 agents ayant chacun 2 actions et 2 observations.

Les tables 7.4 et 7.5 listent les différentes expériences réalisés sur ce problème. Nous avons testé plusieurs heuristiques pour des horizons de 3, 4 ou 5 et pour les différentes fonctions de récompense. Les temps indiqué comprennet le temps de pré-traitement nécessaire pour les différentes heuristiques et si '*' indique un "time-out", ce dernier s'est toujours produit *après* le pré-traitement des heuristiques. Quand il n'y avait pas assez de place en mémoire pour la formulation du problème, nous l'avons indiqué par un '-'.

Nous avons aussi comparé nos algorithmes avec les algorithmes de la littérature, qu'ils soient exacts ou non. Ainsi, dans la table 7.6, E indique un algotithme exact, N un algorithme localement

Programme	Horizon T	Temps (Secs)	Horizon T	Temps (Secs)	Horizon T	Temps (secs)
MILP2	3	0.86	4	900	5	-
MILP2-Low	3	0.93	4	900	5	-
MILP2-Up	3	1.03	4	907	5	-
MILP2-LOC	3	0.84	4	80	5	*
MILP2-LOC-Low	3	0.84	4	120	5	*
MILP2-LOC-Up	3	0.93	4	10.2	5	25

TAB. 7.4 – Temps d'exécutioni du **MILP2** sur le problème MABC.

Programme	Horizon T	Temps (Secs)	Horizon T	Temps (Secs)	Horizon T	Temps (secs)
MILP3	3	0.391	4	3.53	5	-

TAB. 7.5 – Temps d'exécutioni du **MILP3** sur le problème MABC.

exact (équilibre de Nash) et A un algorithme approché. Un blanc indique que l'algorithme n'a pu se terminer, soit par manque de temps soit pas manque de place mémoire.

Ces résultats montrent une nouvelle fois que nos deux algorithmes sont plus rapides que les algorithmes existants. Il faut toutefois noter que le problème MABC est un problème "facile", pour plusieurs raisons :

- 62% des historiques sont localement superflus, ce qui réduit drastiquement la taille du problème.
- il n'y a que 2 actions par agent.
- La valeur optimale du POMDP équivalent est la même que celle du DEC-POMDP. Le plan de coupure qui est en déduit est donc très efficace.

Ces raisons font aussi que le problème MABC n'est pas forcément un problème très caractéristiques des difficultés posées par les DEC-POMDP.

7.5 Problèmes aléatoires

Pour ces problèmes, les fonctions de transition et de récompense ont été générées aléatoirement. Nous donnons ici les résultats pour deux problèmes parmis ceux qui ont été générés, ces

Algorithme	Type	Horizon	Temps	Horizon	Temps	Horizon	Temps
DP	E	3	5 s	4	900 s	5	
MAA*	E	3	< 1 s	4	3 h	5	
Recursive MAA*	E	3	< 1 s	4	1.5 h	5	
PBDP	E	3	< 1 s	4	2 s	5	10^5 s
MILP2	E	3	< 1 s	4	10.2 s	5	25 s
Approx. PBDP	A	3	< 1 s	4	< 1 s	5	10 s
MBDP	A	3	0.01	4	0.01	5	0.02 s

TAB. 7.6 – Runtimes Of Existing Algorithms On The MA-Tiger Problem.

Programme	Temps min (Secs)	Temps Max (secs)	Moyenne	Deviation Std.
MILP2	2.45	455	120.6	183.48
MILP3	6.85	356	86.88	111.56

TAB. 7.7 – Temps d'exécution de **MILP2** et **MILP3** sur le problème à 2 agents **Random1** avec un horizon de 4.

Programme	Temps min (Secs)	Temps Max (secs)	Moyenne	Deviation Std.
MILP2	1.45	10.46	4.95	3.98
MILP3	5.06	12.53	7.28	2.43

TAB. 7.8 – Temps d'exécution de **MILP2** et **MILP3** sur le problème à 2 agents **Random2** avec un horizon de 3.

problèmes sont appelés **Random1** et **Random2**. Random1 a 2 actions et 2 observations par agent tandis que Random2 a 3 actions et 2 observations par agent.

Les tables 7.7 et 7.8 donnent les résultats de nos teests pour des problèmes à 2 agents. Le table 7.9 présente quelques résultats pour des problèmes avec 3 agents avec un horizon de 3, car un horizon de 4 demandait trop de mémoire pour être résolu.

7.6 Bilan

Ce chapitre nous a permis de tester le comportement de nos algorithmes sur plusieurs problèmes souvent évoqués dans la littérature sur les DEC-POMDP. Nous avons aussi généré plusieurs problèmes aléatoires, notamment des problèmes avec 3 agents. Pour des horizons "raisonnables" (de l'ordre de 4), nos programmes mathématiques permettent de résoudre ces problèmes de manière exacte bien plus rapidement que les algorithmes existants.

Ces expériences montrent aussi que les heuristiques jouent un rôle important dans la capacité qu'ont nos programmes à résoudre des DEC-POMDP. Il faut aussi noter que, bien que **MILP3** soit le plus "petit" de nos programmes, il n'est pas forcément le plus efficace car il s'appuie sur de nombreuses variables *entières*.

Programme	Temps min (Secs)	Temps Max (secs)	Moyenne	Deviation Std.
MILP2	21	173	70.6	64.02
MILP2-Low	26	90	53.2	24.2
MILP5	754	2013	1173	715

TAB. 7.9 – Temps d'exécution de **MILP2** et **MILP5** sur le problème à 3 agents **Random2** avec un horizon de 3.

Chapitre 8

Conclusions et perspectives

Une nouvelle fois, ce chapitre est le pendant du Chapitre 8 du manuscrit anglais, chapitre que l'on pourra trouver à la page 165. Les conclusions y sont intégralement évoquée mais les perspectives sont plus longuement détaillée dans le chapitre en anglais.

8.1 Conclusions

Cette thèse s'est intéressée au problème de la planification dans les problèmes décentralisés. On se trouve confronté à ce genre de problèmes quand on cherche à contrôler automatiquement des processus à partir de plusieurs postes de contrôle indépendants. Un tel contrôle décentralisé est nécessaires dans plusieurs application et désirable dans d'autres. On peut trouver des exemples de tels problèmes dans des domaines comme la détection [TA85], de la télécommunication [Ros83], de la multi-robotique [BZLG04], etc.

Mais la planification pour les problèmes décentralisés est difficile d'un point de vue computationel. Cette difficulté a déjà été formalisée et reconnue depuis au moins deux décades. En 1985, Tsitsiklis et Athans [TA85] ont prouvé que le problème de décision en équipe (TDP), un des modèles mathématique de problèmes décentralisés statiques proposé en 1959 par Radner [Rad59], était NP-dur. Le cas général des problèmes décentralisés dynamiques est capturé par les Processus Décisionnels de Markov Partiellement Observables Décentralisés (DEC-POMDP) ou, de manière équivalente, par les problèmes de décision d'équipe markoviens (MTDP), et Bernstein et ses collaborateurs ont prouvé en 2002 que ces modèles sont encore plus difficiles ; ils sont NEXP-complets [BGIZ02].

Ces résultats de complexité qui ne sont pas encourageant ont freiné le développement d'algorithmes de planification pour les problèmes décentralisés, particulièrement pour les problèmes dynamiques. Cependant, les avancées spectaculaires en terme de puissance de calcul au cours des dernière décades font que nous sommes maintenant à une étape où il est envisageable de résoudre des problèmes de petite taille. Dès lors, la recherche dans ce domaine a connu un regain d'intérêt et des algorithmes de planification ont été proposés. Ainsi, des articles publiés depuis 2002 – par exemple [BGIZ02], [CSC02], [NTY+03], [BZLG04], [CRRL04], [SCZ05], [BM06], [SC06], [PZ07a], [ABZ07], [SZ07], etc. – proposent et mettent en œuvre un large éventail de techniques et d'algorithmes pour les problèmes décentralisés.

Pourtant, les alogoritmes exacts qui existent pour les DEC-POMDP semblent tous ce comporter selon le pire scénario possible, même pour des problèmes de petite taille. En théorie, ils nécessitent une place mémoire et/ou un temps d'exécution qui sont doublement exponentiels en l'horizon du problème, et cela semble bien être le cas en pratique.

Cette thèse a cherché à apporter une contribution à ce domaine en proposant des algorithmes qui sont capables de résoudre de manière exacte des DEC-POMDP de petite taille en utilisant relativement moins d'espace mémoire et en un temps réduit. Les algorithmes présentés dans ce manuscrit ont montré que leur temps d'exécution était significativement plus court que ceux des algorithmes existants. De plus, en pire cas, nos algorithmes ont besoin d'une quantité de mémoire qui est exponentielle en la taille de l'horizon et non doublement exponentielle.

De plus, un des effets de bord de nos recherche a permis la conception d'un algorithme pour trouver un équilibre de Nash pour les jeux sous forme extensives avec information imparfaite (ce qui correspond au modèle des Jeux Stochastiques Partiellement Observables ou POSG) où plus de deux agents sont acteurs, dans le cas où ce jeu admet un équilibre pure. A l'heure actuelle, très peu d'algorithmes permettent de résoudre ce type de jeu.

La principale différence entre notre approche et les approches existantes tient dans le fait que nous utilisons une *formulation séquentielle* des politiques plutôt que la représentation classique sous forme d'un arbre. Les avantages intuitifs et théoriques de cette approche ont été détaillé au chapitre 3 et les avantages pratiques ont été illustré par les expériences détaillées dans le chapitre 7. Notre approche a ainsi permis de proposer plusieurs programmes linéaires mixtes entiers à valeurs dans 0-1 (MILP) dont les solutions définissent des politiques jointes optimales. Ainsi, un des points clef de notre succès réside dans la robustesse, l'efficacité et la généralité des solveurs de MILP existants. Nous avons utilisé le solveur vendu par ILOG mais d'autres solveurs existent, comme par exemple le solveur libre NEOS. La robustesse de ces solveurs n'est pas étonnante. En effet, un très grand nombre de problème d'optimisation intéressant l'industrie et les sciences appliquées peuvent se mettre sous la forme de MILP. Notre approche a ainsi profité des avancées faites dans le domaine de la programmation linéaire.

Les expériences réalisées avec nos programmes linéaires révèles trois faits :

- En premier lieu, nos expérimentation montrent clairement que le temps nécessaire pour trouver une politique jointe optimale sous forme séquentielle en résolvant un MILP est moindre, d'un ordre de magnitude de 1 ou 2, que le temps nécessaire en utilisant un algorithme classique et une politique sous forme canonique.
- Deuxièmement, les heuristiques jouent un rôle important dans le temps de résolution de nos MILP. Il est vrai que sans ces heuristiques, notre approche pourrait présenter un avantage moindre comparée aux autres techniques. Mais cela serait oublier le fait que ces autres techniques font elles-aussi appel à des heuristiques (qui sont souvent similaires à celles que nous avons employées) et, sans ces heuristiques, ces méthodes classiques seraient sans doute incapables de résoudre le moindre DEC-POMDP.
- Troisièmement, bien que notre approche soit considérablement plus rapide que les algorithmes existants, seuls des problème de très petites tailles peuvent être résolus (moins de 3 actions par agent, moins de 2 observation, moins de 10 états, un horizon de 5 au plus). Autrement dit, notre approche est une avancée certaine en ce qui concerne le temps de

résolution mais pas en ce qui concerne les exigences des problèmes concernant la taille de la mémoire nécessaire.

Nous pouvons résumer ce travail en disant que les approches actuelles (ce qui inclus les algorithmes présentés dans cette thèse) sont capable de résoudre de manière exacte des DEC-POMDP de petite taille et pour un horizon court. Nous entendons par là qu'il est possible de trouver une politique jointe optimale. La plupart des problèmes pratiques nécessitent cependant un plus grand nombre d'actions, d'obesrvations et d'états. Ces problèmes sont donc encore hors de portée des méthodes de résolution exacte, ce qui est implicite dans le fait que leur complexité soit NEXP-complète.

8.2 Quelques directions pour des travaux futurs

Il nous semble que la principale direction de travail concerne l'application des contributions de cette thèse à ces problèmes où l'horizon est soit grand soit infini.

8.2.1 Problème avec un grand horizon

Une proposition naturelle, bien que naïve et potentiellement largement sous-optimale, pour résoudre les problèmes avec un horizon T très grand est de découper ce problèmes en problème d'horizon beaucoup plus court. Comme nos algorithmes permettent de résoudre ces problème en un temps extrèmement réduit, il est envisageable de découper un long problème en une multitude de problèmes courts.

Imaginons que nous soyons face à un problème avec un horizon $\mathbb{T}$ très long, une façon de procéder pourrait être la suivante, avec T un horizon court et raisonnable.

1. Intialiser k à 1 et β^k à α, l'état initial du DEC-POMDP.
2. Trouver une politique jointe optimale p^k d'horizon T en utilisation une formulation séquentielle et un MILP pour l'état initial β^k (T est très peit, disons 3 ou 4).
3. Déterminer un état $\beta' \in \Delta(S)$ qui est atteint quand p^k est exécutée à partir de β^k.
4. Incrémenter k de1, changer β^k en β'. Si k est plus grand que $\mathbb{T}/T$, alors on arrête, sinon on reprend au point 2.

La difficulté principale consiste à choisir l'état $\beta' \in \Delta(S)$ qui sera le point de départ de nouveau problème d'horizon T. Le manuscrit en anglais présente plusieurs méthode pour choisir ce point parmis tous les points qui peuvent être atteient après l'exécution de la politique p^k. Il est ainsi possible de prendre le plus probable, le plus prometteur, le points moyen, etc.

A l'image de certains algorithmes approchés de la littérature, il est aussi possible de limiter à l'avance les ressources dont disposerait notre algorithme, comme le fontpar exemple [SZ07] ou [SC06]. Cela reviendrait à limiter les historiques support des politiques recherchées et nécessiterait des heuristiques pour choisir ces historiques qui semblerait pertinents *a priori*.

8.2.2 Problèmes à horizon infini

Cette thèse s'est intéressée exclusivement aux problèmes en horizon fini. Il existe néanmoins une classe importante de problèmes où l'horizon est infini. Les critères d'optimalité sont alors différent car le critère que nous avons employé pourrait diverger. On utilise classiquement un critère pondéré ou un critère moyen.

– *critère pondéré* : γ est un réel de $(0, 1)$:

$$E\Big\{\sum_{t=1}^{\infty} \gamma^t R(s^t, (a_1^t, a_2^t, \ldots, a_n^t))\Big\} \tag{8.1}$$

– *critère moyen* :

$$E\Big\{\lim_{T\to\infty} \sum_{t=1}^{T} R(s^t, (a_1^t, a_2^t, \ldots, a_n^t))/T\Big\} \tag{8.2}$$

Dans ces problèmes, la politique optimale est statique et certains auteurs ont donc proposé de chercher des politiques pouvant s'exprimer comme des automates à états finis [BHZ05]. Cette formulation peut conduire à des programmes non-linéaires qui sont proches de ceux que nous avons exposé, notamment du programme **NLP1**. Nous pensons qu'il est possible de trouver une adaptation de notre approche et de nos heuristiques pour chercher des solutions sous la forme de contrôleur stochastiques finis.

Part II

English Manuscript

Chapter 1

Planning For Decentralized Problems

1.1 Decentralized Problems

Planning is central to all our activities. We all make plans, whether it is planning a trip or planning the weekly budget. In industrial and economic activities, planning is a fundamental activity. However, in many problems, the inputs that go into conceiving a plan are far too complex and intricate for us to conceive a plan by hand, and the exercise of conceiving a plan must be automated. Apart from the conception of a plan itself, the execution of a plan is also often required to be automated. For instance, the exploration of a terrain that is dangerous or inaccessible to humans, such as the surface of a planet or a minefield, must be carried out by robots endowed with the capacity to do such exploration. What is then required is to install an exploration plan on the robot that tells it how to act or rather how to react to events that occur during the exploration. The subject of our thesis is the automatic conception of plans that are to be executed in an automatic manner.

If a plan is to be executed in automatic manner, then the plan must be a *complete* one. That is, the plan must cover every contingency the decision maker or *agent* may encounter during the execution of the plan. An agent could be a person, a robot, a machine, a computer program etc. The plan must tell the agent which action to take in any given contingency. The assumption in problems where the execution of a plan is to be automatized is that the agent who executes the plan does not have the capacity to determine the optimal action for a given contingency. Thus, if the plan is a complete one, the agent is thereby not stranded in any contingency, not knowing what action to take therein.

The computational complexity (the space and time required) of conceiving a plan that is to be executed in an automatic manner depends on the nature of the problem. It has been long recognized that the main bifurcation in this regard is between **centralized** problems and **decentralized** problems. A centralized problem requires only one agent for its resolution while a decentralized problem requires multiple agents for its resolution. The exploration of a terrain by a single robot is a centralized problem while the exploration of a terrain by a team of robots, each of whom explores a portion of the terrain, is a decentralized problem. Hence, in a centralized problem, only one plan is required. The outcome of the problem depends on the plan chosen. In a decentralized problem a tuple of plans is required, one plan in the tuple per agent; each plan in the tuple is to be executed by an agent independently of the other agents. The outcome of the problem depends on the tuple of plans chosen.

To take an example of a decentralized problem from Marschak [Mar55] who initiated research into decentralized problems, in a business firm, we have different executives in charge of different departments (production, accounting, advertising etc.). Each executive is required to take decisions regarding his own department without necessarily knowing what decisions other executives take. But the decisions of the executives together determine the performance of the firm. This is a decentralized problem where we are required to conceive a tuple of plans of actions, each plan in the tuple to be used by an executive.

Conceiving an optimal tuple of plans for a decentralized problem is a *much more* difficult task that conceiving an optimal plan for a centralized problem simply because of the combinatorial explosion that occurs in considering all tuples of plans in finding an optimal tuple of plans. In mathematical programming terms, planning for a decentralized problem represents a nonconvex (if minimizing) or nonconcave (if maximizing) nonlinear program whereas planning for a centralized problem represents a linear program. Therefore, decentralized problems are NP-hard or worse (in the general case, they are NEXP-hard) while centralized problems are generally of polynomial complexity.

We shall now take up a few examples of decentralized problems in order to better understand the inherent difficulty in these problems.

1.1.1 The Team Decision Problem

The (discrete) Team Decision Problem (TDP) [Rad59] is a simple mathematical formulation of a decentralized problem. It is an apt example to describe the difficulty involved in conceiving an optimal tuple of plans for a decentralized problem. The 2-agent case of the TDP (generalizable to $n \geq 2$ agents) is defined by the tuple (S_1, S_2, A_1, A_2, R). For each agent i, S_i is the set of contingencies or *states* the agent may find himself in, and A_i is the set of *actions* available to the agent. $R : S_1 \times S_2 \times A_1 \times A_2 \rightarrow \mathbb{R}$ is a *reward function*; if agent 1 takes action $a_1 \in A_1$ when he is in state $s_1 \in S_1$ and agent 2 takes action $a_2 \in A_2$ when he is in state $s_2 \in S_2$, the agents receive the collective reward $R(s_1, a_1, s_2, a_2)$.

A plan or *policy* of agent i is a function from S_i to A_i. An optimal pair of policies in the TDP is a pair (p_1, p_2) that maximizes the total reward, given as,

$$\sum_{s_1 \in S_1} \sum_{s_2 \in S_2} R(s_1, s_2, p_1(s_1), p_2(s_2)) \tag{1.1}$$

An optimal pair of policies for the TDP can be found by solving the following 0-1 integer nonlinear program (INLP):

$$\text{Maximize} \quad \sum_{s_1 \in S_1} \sum_{s_2 \in S_2} \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} R(s_1, s_2, a_1, a_2) x_1(s_1, a_1) x_2(s_2, a_2) \tag{1.2}$$

Subject To,

$$\sum_{a_i \in A_i} x(s_i, a_i) \quad = \quad 1, \quad i = 1, 2, \forall s_i \in S_i \tag{1.3}$$

$$x_i(s_i, a_i) \quad \in \quad \{0, 1\}, \quad i = 1, 2, \forall s_i \in S_i, \forall a_i \in A_i \tag{1.4}$$

This INLP contains one variable $x_i(s_i, a_i)$ for every state-action pair (s_i, a_i) for each agent i. An optimal solution to the program (x_1^*, x_2^*) constitutes an optimal pair of policies for the given TDP.

The program has a nonconcave, nonlinear objective function (1.2). This makes the INLP a problem of constrained nonconcave nonlinear maximization. Since finding an optimal solution to such a problem is NP-hard [Pap80], finding an optimal pair of policies for the TDP is NP-hard. Of course, this is so independent of the algorithm used. Tsitsiklis and Athans [TA85] proved that finding an optimal pair of policies for the TDP is NP-hard.

We can linearize this 0-1 INLP to a 0-1 integer linear program (ILP). The 0-1 ILP would have a linear objective function. However, this would not change the complexity of the problem since finding an optimal solution to a 0-1 ILP is also an NP-hard problem [Pap80].

Now, consider the following centralized problem, an analogue of the TDP, defined as the tuple ($S = S_1 \times S_2$, $A = A_1 \times A_2$, R). The two agents are now required to act as one. A policy for this problem is a function from S to A. An optimal policy is a policy p that maximizes the total reward, given for this problem as,

$$\sum_{s_1 \in S_1} \sum_{s_2 \in S_2} R(s_1, s_2, p(s_1, s_2)) \tag{1.5}$$

An optimal policy for this problem can be found by solving the following linear program (LP):

$$\text{Maximize} \quad \sum_{s_1 \in S_1} \sum_{s_2 \in S_2} \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} R(s_1, s_2, a_1, a_2) x(s_1, s_2, a_1, a_2) \tag{1.6}$$

Subject To,

$$\sum_{a_1 \in A_1} \sum_{a_2 \in A_2} x(s_1, s_2, a_1, a_2) = 1, \quad \forall s_1 \in S_1, \forall s_2 \in S_2 \tag{1.7}$$

$$x(s_1, s_2, a_1, a_2)) \geq 0, \quad \forall s_1 \in S_1, \forall s_2 \in S_2, \forall a_1 \in A_1, \forall a_2 \in A_2 \tag{1.8}$$

The LP contains one variable $x(s_1, s_2, a_1, a_2)$ for every pair of state-action pairs, (s_1, a_1) and (s_2, a_2). An optimal solution x^* to this LP constitutes an optimal policy to this centralized problem. Since solving an LP is a problem of polynomial complexity [Meg87], finding an optimal policy for the centralized TDP problem is of polynomial complexity. Note that in an optimal solution, the value of each variable is either 0 or 1 even if each variable is only constrained to be a non-negative variable. In other words, we do not the variables to be integer variables. This helpful property is on account of the *total unimodularity* of the matrix of coefficients of the constraints (1.7). A totally unimodular matrix (TUM) is a matrix for which every square non-singular submatrix is unimodular. A square matrix is unimodular if its determinant is either 0, 1 or -1.

The TDP can be used to model practical problems of *decentralized detection* (DD) [TA85]. A simple example of a problem of DD is of group of sensors who are required to take measurements of a process (which could be natural phenomena, such as the weather, or mechanical phenomena, such as airplane traffic). Each sensor takes measurements independently of the other sensors, based on its own observations. Each sensor is required to send the measurements it collects to a central agency which collates all received data in order make estimates about the state of the process. Each sensor is thereby required to decide which measurements it should send to the center and which ones it should not, so that the center makes as accurate an estimate of the state as possible.

1.1.2 The Multi-Access Broadcast Channel Problem

The multi-access broadcast channel (MABC) problem [Ros83] is a practical example of a decentralized problem. In this problem, it is required to decide how to allocate a single broadcasting channel amongst a group of n transmission stations $i = 1, 2, \ldots, n$. Each station i has a buffer that can hold a maximum of m_i messages at a time. We imagine time to be split into discrete periods. In a period, only one station can send a message over the channel. If two or more stations send messages over the channel in the same period, a collision of the messages occurs and each of the messages is lost. Messages arrive at the stations at probabilistic rates, independent of one another.

In order to allocate the channel among the stations, we are required to formulate a set of transmission policies, one policy per station. Each transmission policy is to be implemented at a station before the duration begins. The policies need not be identical. The transmission policy of each station i is a function from S_i to A_i where S_i is the set $\{0, 1, 2, \ldots, m_i\}$ and A_i is the set {'use', 'don't use'}. Thus, the policy of a station determines if in a period the station should use the channel or not as a function of the number of messages in its buffer. An optimal tuple of transmission policies for this problem is one which maximizes the *average* number of messages successfully transmitted per period over the channel.

In the n-station case, the MABC represents a nonlinear program (with an objective function of degree n) with separable constraints. For the 2-station case, the corresponding bilinear program with separable constraints can be re-formulated as a 0-1 mixed integer linear program (MILP) as shown in [PZ07a], whose optimal solution yields an optimal pair of transmission policies. This pair of policies maximizes the average number of messages successfully transmitted per period. The mathematical model used in [PZ07a] to formulate the MABC problem was the decentralized Markov decision process or DEC-MDP model (with independence of state transitions), a generalization of the TDP model. The DEC-MDP model is described in detail in Chapter 2, Section (2.1.2). Effectively, a n-agent DEC-MDP (with independence of state transitions) describes an n-tuple of Markov decision processes or MDPs conjoined by a common reward function.

By analogy, the centralized case of the MABC problem can be modeled as an MDP and thereby solved by a linear program. In the centralized case, only one policy is required. Such a policy is a function from $S_1 \times S_2 \times \ldots S_n$ to the set $\{1, 2, \ldots, n\}$. The policy decides which of the n stations should use the channel in a period given the numbers of messages in the buffers of the n stations.

1.1.3 Queue Load Balancing Problem

In the MABC problem, the rates at which messages depart from the stations are independent of one another. This essentially means that the local state of a station is unaffected by the actions of other stations. A generalization of this problem is then the case where a station's state is affected by the actions of other stations. An interesting example of this sort is the problem of queue load balancing [CRRL04]. In this problem, we are given n servers each of whom has its own queue with a certain capacity. Jobs arrive at the servers at rates independent of one another. Time is assumed to be split into discrete time periods, and in a period, a server can service one job with a certain probability.

If the queue of a server is full, and a job still arrives at the server, the job cannot be serviced and is considered as lost. If a server has full queue, it has the option of transferring the job to another server. It is assumed that each server is directly connected to two servers only to which it can transfer a job. Thus, when a job arrives at a server, the server has three alternatives: (K) It can either keep the job in its queue (if it has space), (L) transfer the job to the server on its left or (R) transfer the job to the server on its right. The state of the system in any period is described by the tuple (s_1, s_2, ..., s_n) where s_i is the number of jobs in server i's queue. Thus, even if jobs arrive at the servers at rates independent of one another, the contents of the queue of a server can be modified by the actions of other servers; on the other hand, in the MABC, the contents of the buffer of a station are unaffected by the actions of other stations.

The cost incurred by the system in a time period t is thereby $\sum_{i=1}^{n} s_{i,t}^2$ where $s_{i,t}$ is the number of jobs in server i's queue in period t. Additionally, any job transfer incurs a cost of 2 and any lost job incurs a cost of 50. We are required to formulate a tuple of transfer policies, one policy per server. A transfer policy implemented on a server that tells the server whether to accept an arriving job in its queue, to transfer it to the left server's queue or to transfer it to the right server's queue. The transfer policy of each server i is a function of not only the number of jobs in i's queue but also the number of jobs in the queues of the servers to its left and right. It is thus a function from $S_i \times S_{i+1} \times S_{i-1}$ to {K, L, R}, where $i + 1$ and i - 1 denote the respectively the servers on the left and right to i, and S_i denotes the set {0, 1, 2, ..., q_i}, q_i being the capacity of server i's queue.

An optimal tuple of transfer policies is one which minimizes the *discounted sum of costs* incurred over an infinite horizon. This problem can be modeled as a DEC-MDP but without independence of state transitions. [CRRL04] present a linear program that finds a tuple of policies that is not guaranteed to be optimal but whose cost is only slightly higher within a known error than that of the optimal tuple of policies. The centralized analogue of this problem is similar to the centralized analogue of the MABC. In the centralized case of the problem, we are required to conceive only one transfer policy, a function from $S_1 \times S_2 \times \ldots S_n$ to the set {1, 2, ..., n}. The policy decides to which server's queue an arriving job must be directed, given the numbers of messages in the queues of the servers. Unlike the decentralized case, for the centralized case, we can conceive a linear program that find is guaranteed to find an optimal policy.

1.1.4 Two-Machine Maintenance Problem

This problem is adapted from the wellknown one-machine maintenance problem [SS73], which is a centralized problem and which can be formulated as a partially observable MDP (POMDP). In the two-machine example, we are given two machines M1 and M2 that convert a certain raw material into a finished product. The raw material is processed by both machines.

The two machines M1 and M2 are operated and managed by two different crews. At the beginning of each day, each crew is required to ascertain the state of the machine it manages in order to know the advisability of operating it on that day. The possible states of a machine are Operable (O) and Due For Serviced (DS). A crew has two alternatives to ascertain the state of a machine: (S) Open the machine, inspect its parts and service them if required or (QC) examine the quality of the previous day's lot from the quality control log. The quality of a lot is either High (H) or Low (L).

Alternative S would give a crew a fairly good idea about the state of the machine since it is a function (albeit, a probabilistic one) of the states of its internal parts. Alternative QC is an indirect way of estimating the machine's state since the crew does not get a chance to look at the internals of the machine. The quality of a lot is a function of the states of the machines. However, alternative S would also cause a loss in productivity since dissembling and assembling a machine requires some time, and production would have to be halted for that time. Alternative Q, while less reliable, has the advantage of not causing a halt in the production. On the other hand, operating a machine when it is due for servicing, may lead to an even greater loss in productivity (for instance, the machine may breakdown, in which case the production might have to be stopped for even more time, or the quality of the good produced may be low).

We are required to conceive a pair maintenance policies. Each policy is for a given period, say a week. Let W be the set {Mon, Tue, ..., Fri}. The policy of each crew tells it which alternative to choose as a function of the quality of the previous day's lot and as a function of the day of the week. Hence, the policy of a crew is a function from {H, L} $\times$ W to {S, QC}. Our objective is to conceive a pair of policies that minimizes the loss in productivity.

Notice that this problem is different from the MABC problem and the queue load balancing problem in that not only does each crew not know the state of the machine it is not responsible for, but it does not even know the state of its own machine since even if it chooses alternative S, it would only be able (in general) to come up with a probability distribution over the set {O, DS}. Therefore, problems such as the two-machine maintenance problem must be modeled using the DEC-POMDP model, a generalization of the DEC-MDP model. The DEC-POMDP model is described in Chapter 2, Section (2.1).

The formulation of three problems described above - MABC, queue load balance and two-machine maintenance - as respectively a DEC-MDP with independence of state transitions, a DEC-MDP without independence of state transitions and as a DEC-POMDP is given in Chapter 2, Section (2.1.3).

1.2 Background

We may identify two distinguishing features of decentralized problems: *partial information of state* and *non-identical information of state*. During the course of the execution of his plan, an agent does not have full information about the state of the problem. In addition, the agents do not have identical information about the state of the problem. In the 2-agent TDP for instance, at any time t, when taking an action, each agent i only knows that he is taking an action in some state s_i^t from S_i; he does not know what state the other agent is in when he is taking an action. Since the state of the problem at time t is described by the pair (s_1^t, s_2^t), each agent has only partial and non-identical information about the state. Similarly, in the MABC problem, the state of the problem at time t is described by the tuple $(s_1^t, s_2^t, \ldots, s_n^t)$ where s_i^t is the number of messages in the buffer of station i; however, since each station has access only to its own buffer, the station does not have complete knowledge about the state of the problem.

Decentralized problems - in one formalism or another - have been the subject of different disciplines such as Game Theory, Operations Research, Control Theory, Artificial Intelligence

etc over the past five decades and more. Different mathematical models are therefore available to formulate such problems:

The TDP Model: Marschak laid the foundation for research in decentralized problems in a seminal paper '*Elements For A Theory Of Teams*' [Mar55]. Radner [Rad59] proposed an early linear programming approach to the problem. Marschak and Radner formulated decentralized problems using the TDP model. The TDP model, however, can be used only to formulate a special case of decentralized problems. It can be used to formulate problems where an action of an agent in a given state has no influence on the states encountered by the agent himself or by the other agents in the future. Thereby, the notion of temporality or sequentiality is not adequately captured in a TDP. In other words, the TDP can be used to model *static* decentralized problems.

The MTDP Model: Decentralized problems such as the MABC problem that require a generalization of the TDP model were formulated as dynamical (Markovian) systems in the 1970s in the domain of control theory [AM80], [Ros83], [Haj84]. However, models that generalize the TDP were formalized and studied only recently. The TDP model was generalized to the Markov Team Decision Problem (MTDP) model [PT02]. The basic version of a 2-agent MTDP can be described as the tuple (S_1, S_2, A_1, A_2, R, $\mathbb{P}$) where the first four elements are as in a 2-agent TDP and $\mathbb{P}$ is a state transition probability function from $S \times A \times S$ to $\mathbb{R}$, where $S = S_1 \times S_2$ and $A = A_1 \times A_2$. $\mathbb{P}(s_1, s_2, a_1, a_2, s'_1, s'_2)$ is the probability that each agent i moves to state $s'_i \in S_i$ if each agent i takes action $a_i \in A_i$ when in state $s_i \in S_i$.

The DEC-POMDP Model: Another mathematical model that can be used to model decentralized problems is the DEC-POMDP model [BGIZ02]. The two models, MTDP and DEC-POMDP, are actually identical. Whereas an MTDP is a natural extension of the TDP, the DEC-POMDP can be seen as a natural extension of the wellknown Partially Observable Markov Decision Process or POMDP model [SS73], used for modeling centralized problems. While the MTDP extends a model for decentralized problems (the TDP) by introducing Markovian dynamics into it, the DEC-POMDP extends a model for centralized problems (the POMDP) by introducing multi-agent elements into it. The DEC-POMDP/MTDP model allows formulating practical problems such as the MABC problem, multi-rover exploration [BZLG04], queue load balancing [CRRL04], sensor networks [NVTY05] etc.

The MAID Model: The Multi-Agent Influence Diagram or MAID model [KM03] is a synthesis of Bayesian Networks and Influence Diagrams. A MAID often results in a very succinct representation of a decentralized problem.

Extensive Game Model: Decentralized problems have also been widely studied in Game Theory as *extensive games*. Extensive games were formalized by Kuhn in the early 1950s [Kuh50]. An extensive game describes a decentralized problem as a tree formed by the agents' actions. Thus, an extensive game explicitly describes all the events the occur over the course of the problem. The other models are in comparison more compact. A finite horizon DEC-POMDP is essentially equivalent to an extensive game with *imperfect information* and identical interests.

The model we shall use in the thesis to study decentralized problems is the DEC-POMDP/MTDP model (henceforth, we shall drop the '/MTDP').

1.3 Complexity

As stated earlier, decentralized problems have a much higher computational complexity that their centralized counterparts. We have seen that while the TDP is an NP-hard problem, the centralized analogue of the TDP is P-complete. The generalization of the TDP to the DEC-POMDP/MTDP results into a problem of even higher complexity.

Whereas finding an optimal policy a POMDP is PSPACE-complete [PT87], [BGIZ02] proved that finding an optimal tuple of policies (henceforth, a joint policy) in a DEC-POMDP is NEXP-hard. Concretely, these complexity results imply that while (in the worst case), to find an optimal policy of a finite horizon POMDP requires time that is exponential in the horizon, (in the worst case), to find an optimal joint policy of a finite horizon DEC-POMDP requires time that is *doubly exponential* in the horizon. [BGIZ02] also showed that a DEC-MDP, a special case of the DEC-POMDP, is also NEXP-hard. However, another special case of a DEC-POMDP, a DEC-MDP with independence of state transitions is a simpler problem; it is only NP-complete [BZLG04]. This special case results into a model that is similar to a TDP. As stated above, the MABC problem can be modeled as this special case.

Similar complexity results from Game Theory are also known. Extensive games with *perfect information* can be solved in time that is linear in the size of the game tree (using the minimax algorithm, for instance). Such games are equivalent to POMDPs. On the other hand, extensive games with imperfect information are much harder to solve. [KM92] proved that finding a sample Nash equilibrium of an extensive game with imperfect information and identical interests is NP-hard. Such games are, as stated before, equivalent to DEC-POMDPs. Note that this hardness result is expressed in the size of the game tree, and not in the depth (horizon) of the game tree. So, it means that solving an extensive game with imperfect information requires time that is exponential in the size of the game tree, or equivalently, solving an extensive game with imperfect information requires time that is doubly exponential in the depth of the game tree.

Decentralized problems modeled as DEC-POMDPs (or its special cases) can be divided into two categories:

Finite Horizon Problems. In such problems, the duration (counted in number of time periods) is finite. The objective in such problems is to maximize the (expected) sum of rewards obtained for the given number of periods (alternatively, it is to minimize the (expected) sum of costs incurred for the given number of period).

Infinite Horizon Problems. Such problems do not have a duration; they are perpetual. The objective in such problems is to either maximize the (expected) discounted sum of rewards obtained or to maximize the (expected) average reward per time period (alternatively, to minimize the (expected) discounted sum of costs incurred or to minimize the (expected) average cost per time period).

The two examples given previously - MABC and queue load balancing - were both presented as infinite horizon problems. If the number of periods is limited to a finite number in both problems, they become finite horizon problems.

Due to the high complexity of solving a DEC-POMDP, there is a paucity of exact algorithms for the problem. An exact algorithm is one which finds an optimal joint policy. Quite some research was done in solving decentralized problems, especially in the domain of control theory in the 1970s and the 1980s. However, that research does not seem to have resulted in generalized algorithms for solving DEC-POMDPs. Exact non-trivial algorithms for DEC-POMDPs have appeared only in recent times. To our knowledge, only a few exact algorithms for the problem have been conceived till date. In the worst case, these algorithms either require space that is doubly exponential in the horizon or time that is doubly exponential in the horizon, or both.

1.4 Contributions Of The Thesis

The principle focus of our thesis is on finite horizon decentralized problems. In the thesis, we study the optimal finite horizon control of a DEC-POMDP. We present new, efficient algorithms for finding an optimal finite horizon joint policy for a given DEC-POMDP

There are two important ways in which our approach differs from existing approaches. The first is that we use a representation of a finite horizon policy that is quite a bit different than that used by existing approaches (which we shall refer to as the canonical form). The second is that, due to this different representation of a policy, we use mathematical programming rather than dynamic programming to find an optimal joint policy.

The mathematical programming approach offers advantages of both space and time. The different mathematical programs presented in this thesis exhibit different sort of performance on sample DEC-POMDPs, but the common feature of these programs is that they constitute *fast algorithms* for finite horizon DEC-POMDPs. In practice, the time taken to solve sample DEC-POMDPs using our mathematical programs is lesser by a magnitude of an order or two (depending on the specific problem) than the time taken by existing exact algorithms. The runtime of the programs is in practice found to be only *exponential* in the horizon whereas existing algorithms seem to require in practice time that is *doubly exponential* in the horizon. Moreover, the space required by these programs is only exponential in the horizon whereas some of the existing algorithms require space that is doubly exponential in the horizon.

The different representation of a policy we adopt is called a *sequence-form* of a policy. This form of a policy was introduced by Koller, Megiddo and von Stengel (KMvS) - [KMvS94], [KM96], [vS96] - for solving extensive games with imperfect information. The KMvS approach represents a major breakthrough in efficiently solving extensive games with imperfect information. By using the sequence-form of the policy, KMvS showed how an extensive game can be solved in space that is linear in the size of the game; the use of the canonical form of a policy requires space that is exponential in the size of the game. Due to the reduction in space, not only can games that are out of the reach of previous algorithms be solved, but the time taken to solve a game is significantly reduced. The KMvS approach results in fast algorithms for extensive form games with imperfect information. As acknowledged by von Stengel [vS02], an approach similar to the KMvS approach, including the use of the sequence-form of a policy, was independently proposed decades before them in Soviet literature [Rom62].

Despite the evident efficiency of the use of the sequence-form of a policy for extensive games, the use of this form has not yet attracted much attention for solving DEC-POMDPs. This is

not surprising in itself since the use of the sequence-form essentially entails using the KMvS approach, and the applicability of the approach to DEC-POMDPs is not evident. The KMvS approach consists of formulating an extensive-form game as a *linear complementarity problem* (LCP) [Mur88]. An LCP is a mathematical program which does not have an objective function, and which consists of a set of linear equations, linear inequalities and complementarity constraints. An LCP can be solved by the Lemke-Howson Algorithm [LH64] or the similar Lemke's complementary pivoting algorithm [Lem65]. A solution of the LCP generated by the KMvS approach is a *Nash Equilibrium*[Nas51] of the game. The adaptation of this approach to solving DEC-POMDPs poses two serious challenges.

(i) First, since the KMvS approach is meant for solving extensive games, it only finds a sample Nash Equilibrium of the game. Therefore, if the approach is applied as it is to solving a DEC-POMDP, it would only find a Nash Equilibrium of the DEC-POMDP. A Nash Equilibrium of a DEC-POMDP is a *locally* optimal joint policy. In other words, it is a local maximum point of a (nonconcave) nonlinear function. A Nash Equilibrium is not a satisfactory solution to a DEC-POMDP because the difference in the value of an optimal joint policy and a locally optimal joint policy can be arbitrarily high. In essence, finding a locally optimal joint policy of a DEC-POMDP may not be much better than just selecting a joint policy at random. Hence, the KMvS approach would have to be modified to find an optimal joint policy. This modification is non-trivial, to say the least.

(ii) Second, the KMvS approach is for 2-player games, which means that if it is used as it is for solving DEC-POMDPs, it can only be used to find a Nash Equilibrium of a 2-agent DEC-POMDPs. It may be possible to extend the KMvS approach to solve 3-or-more-player extensive games using the Govindan-Wilson algorithm [GW01] but such an application does not seem to have been attempted yet.

The thesis solves both these challenges by proposing new mathematical programs instead of an LCP. The challenges are met through two different approaches using different properties of a DEC-POMDP. As stated above, both approaches are based on the use of the sequence-form of a policy.

A Combinatorial Optimization Approach: In this approach, we treat the problem of finding an optimal joint policy of a DEC-POMDP as an instance of combinatorial optimization. Wellknown examples of combinatorial optimization include the TDP, the traveling salesman problem (TSP), the quadratic assignment problem (QAP) [PS82] etc. Typically, such problems involve finding a subset of a given finite set that satisfies some criteria. A policy in the sequence-form (described in detail in Chapter 3) is essentially a subset of the set of *histories*. Hence, the problem of finding an optimal joint policy can be treated as an instance of combinatorial optimization. We show how some of the linearization techniques of this domain can be directed to finding an optimal joint policy. Through this approach, we conceive two 0-1 mixed integer linear program (MILP) each of whose solutions yields an optimal joint policy of the given n-agent DEC-POMDP, $n \geq 2$.

An Optimal Nash Equilibrium Search Approach: This approach is based on the principle that every optimal joint policy is *also* a Nash Equilibrium. The approach therefore hews closely to the KMvS approach to a point. To be precise, in this approach, we conceive an LCP à la KMvS. Since the LCP is not adequate, we transform the LCP to an equivalent 0-1 MILP. A

solution to this 0-1 MILP is a optimal joint policy or in other words, an *optimal* Nash Equilibrium, of the given DEC-POMDP. We present two versions of this 0-1 MILP, one for the 2-agent case and a different one for the 3-or-more agents case.

The two 0-1 MILPs presented in the second approach also find an applicability in *partially observable stochastic games* (POSGs). The POSG model is the competitive analogue of the DEC-POMDP model; it is used to formulate problems of competition while the DEC-POMDP model is used to formulate problems of cooperation. In essence, in a POSG the agents do not necessarily have the same reward or cost function. A Nash Equilibrium of a 2-agent POSG can be found by adapting the 0-1 MILP for a 2-agent DEC-POMDP. It can also be found by the KMvS approach. However, while the latter is only capable of finding a sample Nash Equilibrium, the 0-1 MILP can be used to find a Nash Equilibrium with desired properties such as one which maximizes the sum of rewards of the two agents.

The 0-1 MILP for the 3-or-more agents case (which is again presented in two different formulations) can be used to find a Nash Equilibrium for a POSG with 3 or more players provided that the POSG has a *pure* Nash Equilibrium (this is described in Appendix (A)). POSGs or extensive games which have a pure Nash Equilibrium are rare. However, the 0-1 MILP can be useful because even for POSGs/games that do have a pure Nash Equilibrium, apart from the Govindan-Wilson algorithm, no other efficient algorithm is known for finding such a Nash Equilibrium.

Heuristics: The other important contribution of this thesis is that we introduce heuristics for boosting the performance of the mathematical programs we propose. These heuristics take advantage of the succinctness of the DEC-POMDP model and enable us to reduce the size of the mathematical programs (resulting also in reducing the time taken to solve them). The heuristics constitute an important pre-processing step in solving the programs. We present two types of heuristics: the elimination of *extraneous* histories (this reduces the size of a program) and the introduction of *cuts* in the program (this reduces the time taken to solve a program).

Finally, while the focus of our thesis is on finite horizon DEC-POMDPs, it may be possible to adapt our approach to the infinite horizon case. To be precise, it may be possible to conceive an infinite horizon policy in the sequence form, and use mathematical programming to find an optimal infinite horizon joint policy (in the sequence form). The performance of our mathematical programs for the finite horizon case gives us hope that such a mathematical program may also prove to be a fast algorithm and scale to larger problems. We elaborate on this possibility in the thesis, in Chapter 8.

1.5 Summary

In this chapter we have described decentralized problems and planning for such problems. We have given examples of decentralized problems and stated in brief some of the mathematical models that enable the formulation of such problems. We have described the computational complexity of planning for decentralized problems. We have outlined the contributions of this thesis. The remainder of the thesis is organized as follows.

Chapters 2 and 3 set-up the preliminaries. In Chapter 2, we describe in detail the DEC-POMDP model. This is the model we shall use for formulating decentralized problems. We also describe existing algorithms that find an optimal joint policy of a finite horizon DEC-POMDP. In Chapter 3, we describe the sequence-form representation of a finite horizon policy in a DEC-POMDP. We also describe a (nonconcave) nonlinear program whose optimal solution is an optimal finite horizon joint policy in the sequence-form.

Chapters 4, 5 and 6 constitute the main body of the thesis. In Chapter 4, we present 0-1 mixed integer linear programs (MILPs) based on a combinatorial optimization approach, each of whose solutions is a optimal finite horizon joint policy. In Chapter 5, we present 0-1 MILPs based on an approach to find an optimal Nash Equilibrium. In Chapter 6, we presents heuristics to improve the time and space required to solve these MILPs.

In Chapter 7, we present a comparison of the 0-1 MILPs presented in the thesis and their computational experience on different problems formulated as finite horizon DEC-POMDPs. Finally in Chapter 8, we summarize the thesis and point out some directions for future work.

Chapter 2

Decentralized POMDPs

2.1 The DEC-POMDP Model

In modeling a decentralized problem as a decentralized partially observable Markov decision process (DEC-POMDP), we characterize it as the decentralized control of a discrete-state, discrete-time, Markov process by n agents, $n \geq 2$, whose state is completely hidden from the agents. The Markov process unfolds over discrete time periods. In each period, it assumes a state. But instead of observing the state, each agent receives partial information about the state of the process in the form of an *observation*. An observation received by an agent in a period is a piece of information that is probabilistically related to the state of the Markov process in that period.

A **DEC-POMDP** is defined as the tuple $\mathcal{D} = (I, S, \{A_i\}, \mathbb{P}, \{O_i\}, \mathbb{G}, R, \alpha)$:

- $I = \{1, 2, \ldots, n\}$ is a set of *agents*.
- S is a finite set of *states*. The set of probability distributions over S shall be denoted by $\Delta(S)$. Members of $\Delta(S)$ - S shall also be called states. Thereby, members of S shall be also referred to as *corner* states.
- For each agent $i \in I$, A_i is a set of *actions*. $A = \times_{i \in I} A_i$ denotes the set of joint actions.
- $\mathbb{P} : S \times A \times S \rightarrow [0, 1]$ is the *state transition function*. For each $s, s' \in S$ and for each $a \in A$, $\mathbb{P}(s, a, s')$ is the probability that the state of the problem in a period is s' if in the previous period, its state was s and if the agents took the joint action a. Thus, for any time period $t \geq 2$, for each pair of states $s, s' \in S$ and for each joint action $a \in A$, there holds,

  $$\mathbb{P}(s, a, s') \quad = \quad \text{Prob.}(s^t = s' | s^{t-1} = s, a^t = a) \tag{2.1}$$

  Thus, $(S, A, \mathbb{P})$ defines a discrete-state, discrete-time Markov process.
- For each agent $i \in I$, O_i is a set of *observations*. $O = \times_{i \in I} Oi$ denotes the set of joint observations.
- $\mathbb{G} : A \times S \times O \rightarrow [0, 1]$ is a *joint observation function*. For each for each $a \in A$, for each $o \in O$ and for each $s \in S$, $\mathbb{G}(a, s, o)$ is the probability that the agents receive the joint observation o (that is, each agent i receives the observation o_i) if the state of the problem in that period is s and if in the previous period the agents took the joint action a. Thus,

for any time period $t \geq 2$, for each joint action $a \in A$, for each state $s \in S$ and for each joint observation $o \in O$, there holds,

$$\mathbb{G}(a, s, o) \quad = \quad \text{Prob.}(o^t = o | s^t = s, a^{t-1} = a) \tag{2.2}$$

- $R : S \times A \rightarrow \mathbb{R}$ is the *reward function*. For each $s \in S$ and for each $a \in A$, $R(s, a) \in \mathbb{R}$ is the reward obtained by the agents if they take the joint action a when the state of the process is s.
- $\alpha \in \Delta(S)$ is the *initial state* of the DEC-POMDP. For each $s \in S$, $\alpha(s)$ denotes the probability that the state of the problem in the first period is s.

The control of a DEC-POMDP by the n agents unfolds over discrete time periods, $t = 1, 2, \ldots$, as follows. In each period t, the process assumes a state denoted by s_t from S. In the first period, it is chosen according to α. In each period t, each agent $i \in I$ takes an action denoted by a_i^t from A_i according to the agent's *policy*. When the agents take the joint action $a^t = (a_1^t, a_2^t, \ldots, a_n^t)$, the following events occur:

1. The agents obtain the reward $R(s^t, a^t)$.
2. The period changes from t to $t + 1$.
3. The state s^t that the process assumes is determined according by the function $\mathbb{P}$ with arguments s^{t-1} and a^{t-1}.
4. Each agent $i \in I$ receives an observation o_i^t from O_i. The joint observation $o^t = (o_1^t, o_2^t, \ldots, o_n^t)$ is determined by the function $\mathbb{G}$ with arguments s^t and a^{t-1}.

For agent $i \in I$, let $\overline{O}_i^t$ denote the set of sequences of t observations, $t \geq 0$, that can be formed from O_i. $\overline{O}_i^0$ contains only the empty sequence $\varnothing$. In the first period, an agent is said to receive this sequence of observations. A **T-period policy** of agent i is a function π that assigns, to each integer $t = 0$ to T - 1 and to each sequence of k observations $\overline{o} \in \overline{O}_i^t$ an action $\pi(\overline{o}) \in A_i$. In using π, the agent must take action $\pi(\overline{o})$ if the sequence of observations he has received till a period is $\overline{o}$. A **T-period joint policy** $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is an n-tuple of T-period policies where for $i \in I$, σ_i is a T-period policy of agent i. In the next chapter we shall give a different definition of a policy. Hence, a policy according to the above definition shall also be called a policy in the **canonical form** and a joint policy composed of policies in the canonical form shall be called a joint policy in the canonical form.

2.1.1 The Finite Horizon Problem

As stated in Chapter 1, the focus of this thesis is finite horizon DEC-POMDPs. The **Finite Horizon DEC-POMDP Problem** is defined as follows. We are given a DEC-POMDP $\mathcal{D}$ and an integer $T \geq 1$ called the **horizon** of the problem, and our objective is to conceive a T-period joint policy that maximizes the expectation of the sum of rewards collected over T periods given that the state in the first period is chosen according to α. That is, we are required to conceive a T-period joint policy $\sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_n^*)$ that maximizes

$$E\Big\{ \sum_{t=1}^{T} R(s^t, (\sigma_1^*(\overline{o}_1^t), \sigma_2^*(\overline{o}_2^t), \ldots, \sigma_n^*(\overline{o}_n^t)) \Big\} \tag{2.3}$$

where s^t is the state of the Markov process in period t and $\overline{o}_i^t$ is the sequence of observations received by agent i till period t. σ^* is called an **optimal joint policy**.

For a T-period joint policy $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, the expectation

$$E\Big\{\sum_{t=1}^{T} R(s^t, (\sigma_1(\overline{o}_1^t), \sigma_2(\overline{o}_2^t), \ldots, \sigma_n(\overline{o}_n^t))\Big\} \tag{2.4}$$

given that the state in the first period is chosen according to α is called the **value** of σ at α . It shall be denoted by $V(\alpha, \sigma)$. In order to define it, we shall require the following definitions.

Given a state $\beta \in \Delta(S)$, a joint action $a \in A$ and a joint observation $o \in O$, let $\mathcal{T}(o|\beta, a)$ denote the probability that the agents receive joint observation o if they take joint action a in a period t in which the state is chosen according to β is defined as,

$$\mathcal{T}(o|\beta, a) \quad = \quad \sum_{s\in S} \beta(s) \sum_{s'\in S} \mathbb{P}(s, a, s')\mathbb{G}(a, s', o) \tag{2.5}$$

Given a state $\beta \in \Delta(S)$, a joint action $a \in A$, a joint observation $o \in O$, the **updated state** $\beta' \in \Delta(S)$ of β with respect to a and o is defined as (for each $s' \in S$),

$$\beta'(s') \quad = \quad \frac{\sum_{s\in S} \beta(s)\mathbb{P}(s, a, s')\mathbb{G}(a, s', o)}{\mathcal{T}(o|\beta, a)} \tag{2.6}$$

If $\mathcal{T}(o|\beta, a) = 0$, we can let β' be an arbitrary member of $\Delta(S)$.

Given a state $\beta \in \Delta(S)$ and a joint action $a \in A$, $R(\beta, a)$ denotes $\sum_{s\in S} \beta(s)R(s, a)$.

Using the above definitions and notations, the value $V(\alpha, \sigma)$ of σ is defined recursively as follows,

$$V(\alpha, \sigma) \quad = \quad V(\alpha, \sigma, \varnothing) \tag{2.7}$$

where the elements of recursion are (2.8), (2.9) and (2.10) given as follows:

$$V(\alpha, \sigma, \varnothing) \quad = \quad R(\alpha, \sigma(\varnothing)) + \sum_{o\in O} \mathcal{T}(o|\alpha, \sigma(\varnothing))V(\alpha', \sigma, o) \tag{2.8}$$

$\sigma(\varnothing)$ denotes the joint action $(\sigma_1(\varnothing), \sigma_2(\varnothing), \ldots, \sigma_n(\varnothing))$ and α' denotes the updated state of α given $\sigma(\varnothing)$ and the joint observation $(\varnothing, \varnothing, \ldots, \varnothing)$.

For any $\alpha' \in \Delta(S)$, for each $t = 1$ to T - 2, for each tuple of sequences of t observations $\overline{o} = (\overline{o}_1, \overline{o}_2, \ldots, \overline{o}_n)$ where $o_i \in \overline{O}_i^t$ is a sequence of t observations of agent $i \in I$:

$$V(\alpha', \sigma, \overline{o}) \quad = \quad R(\alpha', \sigma(\overline{o})) + \sum_{o\in O} \mathcal{T}(o|\alpha', \sigma(\overline{o}))V(\alpha'', \sigma, \overline{o}o) \tag{2.9}$$

α'' is the updated state of α' given the joint action $\sigma(\overline{o})$ and joint observation $o = (o_1, o_2, \ldots, o_n)$ and $\overline{o}o$ is the tuple of sequences of $(t + 1)$ observations $(\overline{o}_1o_1, \overline{o}_2o_2, \ldots, \overline{o}_no_n)$ where $\overline{o}_i \in \overline{O}_i^t$.

For any $\alpha' \in \Delta(S)$, for each tuple of sequences of (T - 1) observations $\overline{o} = (\overline{o}_1, \overline{o}_2, \ldots, \overline{o}_n)$:

$$V(\alpha', \sigma, \overline{o}) = \sum_{s \in S} \alpha'(s) R(s, \sigma(\overline{o})) \quad (2.10)$$

If $T = 1$, only (2.10) is applicable (with $\alpha' = \alpha$ and $\overline{o} = \varnothing$). If $T = 2$, only (2.8) and (2.10) are applicable. When $T > 2$, all three (2.8) , (2.9) and (2.10) are applicable.

Thus, an optimal joint policy is a joint policy whose value is maximum. The problem of finding an optimal joint policy is NEXP-hard [BGIZ02]. The reason for this high complexity can be partly understood by the fact that the number of T-period joint policies is doubly exponential in T besides being exponential in n (a T-period policy of agent i assigns an action to every sequence of observations of length less than T. The number of observation sequences of length t of agent i is $|O_i|^t$. Thereby, the number of observation sequences of lengths less than T of agent i is $\sum_{t=0}^{T-1} |O_i|^t$. Therefore, the number of T-period policies of agent i is $|A_i|^{\sum_{t=0}^{T-1} |O_i|^t}$ which equals $|A_i|^{\frac{|O_i|^T - 1}{|O_i| - 1}}$).

2.1.2 Special Cases

The DEC-POMDP model is a very general model for formulating decentralized problems. Many practical problems do not require the full modeling capabilities of a DEC-POMDP and can be modeled as special cases of a DEC-POMDP. Two interesting special cases of a DEC-POMDP in this regard are a DEC-MDP and a DEC-MDP with independence of state transitions.

A **DEC-MDP** or Decentralized Markov Decision Process is defined as the tuple (I, $\{S_i\}$, $\{A_i\}$, $\mathbb{P}$, R, $\{\alpha_i\}$). I, A_i, α are as in a DEC-POMDP. For each agent $i \in I$, S_i is a set of *local* states. S denotes the set $\times_{i \in I} S_i$ of states of the underlying Markov process. Thereafter, $\mathbb{P}$ and R are as defined in a DEC-POMDP. The tuple (S, A, $\mathbb{P}$) defines a Markov process. The control of a DEC-MDP unfolds over discrete periods as follows. In each period t, the process assumes a state $s^t = (s_1^t, s_2^t, \ldots, s_n^t)$ from S. However, each agent i only observes s_i^t. Upon taking a joint action a^t in period t, each agent i's local state changes from s_i^t to s_i^{t+1} according to the function $\mathbb{P}$.

A DEC-MDP is not a simpler problem than a DEC-POMDP as proved in [BGIZ02] . It is also an NEXP-hard problem. However, a DEC-MDP with independence of state transitions is a simpler problem than a DEC-POMDP and a DEC-MDP. A DEC-MDP with independence of state transitions is defined as the tuple (I, $\{S_i\}$, $\{A_i\}$, $\{\mathbb{P}_i\}$, R, $\{\alpha_i\}$). Here, I, S_i, A_i, R and α are as defined in a DEC-MDP. However, each agent i has a separate state transition probability function, $\mathbb{P}_i : S_i \times A_i \times S_i \rightarrow [0, 1]$. Thus, the local state of an agent in a period is determined only by the local state of the agent in the previous period and the action taken by the agent in the previous period. It does not depend on the local states and the actions of the other agents. This fact renders the problem much simpler (relative to a DEC-MDP and a DEC-POMDP). A DEC-MDP with independence of state transitions is NP-hard [BZLG04].

2.1.3 Formulating Practical Problems

In Chapter 1, we described three decentralized problems - the MABC problem, the queue load balancing problem and the two-machine maintenance problem. We describe below how these problems can be modeled as DEC-POMDPs or DEC-MDPs.

The MABC problem can be modeled as a DEC-MDP [PZ07a] with independence of state transitions. When modeled as such a DEC-MDP, the corresponding elements of the DEC-MDP are as follows. Each station is considered as an agent. So I is the set $\{1, 2, \ldots, n\}$. For each station $i \in I$, the set of local states S_i is $\{0, 1, 2, \ldots, m_i\}$ where m_i is the maximum number of messages that the station can hold , the set of available control alternatives or actions A_i is {D, U}, where D means don't use and U means use, $\mathbb{P}_i$ defines the rate at which the buffer of station i refills (if empty) and $\alpha_i \in S_i$ defines the initial local state. The reward function is essentially a negative cost function. It can be defined in different meaningful ways. A simple definition is as follows. Each message successfully sent incurs zero cost (therefore a reward of 0). Each message lost incurs a cost of 1 (therefore, a reward of -1).

The queue load balancing problem can be modeled as a DEC-MDP but without independence of state transitions. When modeled as a DEC-MDP, the elements of the DEC-MDP are as follows. Each server is an agent. The set S_i of local states of agent i is the set $\{0, 1, \ldots, q_i\}$, q_i being the capacity of the queue of server i. The set of actions of each agent is {K, L, R}. The state transition function $\mathbb{P}$ models the following events: (i) Jobs arrive at the servers at fixed but independent rates. (ii) A job transfered to a queue that is not filled to capacity increases the number of jobs in that queue by 1 and decreases the number of jobs in the queue of origin by 1. (iii) A job transferred to a queue that is filled is lost. Therefore, only the number of jobs in the queue of origin decreases by 1. As in the MABC, the reward function is actually a cost function. The reward at the end of a period (irrespective of the actions chosen by the agents) is the negative of $\sum_{i=1}^{n} s_{i,t}^2$, where $s_{i,t}$ is the number of jobs in queue i remaining in at the end of that period. Similarly, the actions L and R incur a reward of -2 and the any lost job incurs a reward of -50.

The two-machine maintenance problem must be modeled as a DEC-POMDP. When modeled as a DEC-POMDP, the elements of the DEC-POMDP are as follows. Each crew is an agent. A state of the problem describes the conditions of the two machines. Therefore, the set S of states is {(O,O), (O, DS), (DS, O), (DS, DS)}. The set A_i of each agent i is {S, QC}. The set O_i of each agent i is {H, L}. The state transition function $\mathbb{P}$ must model the following facts: (i) Servicing a machine that is due to service (state DS) renders the machine operable (state O), but with a probability that is less than 1. Servicing one machine does not change the state of the other machine (ii) The action QC taken by either agent degrades the states of both machines. In other words, any joint action in which either one or both actions are QC increases the probability that the state of the problem is (DS, DS). The joint observation function $\mathbb{G}$ must map the state of a machine to the quality of the lot produced. The quality of the previous day's lot must be a good, if not accurate indicator of the states of the machines on a day. We can conceive different types of reward function for this problem. A simple reward function is as follows. The cost of servicing a machine that is due for service is considered as 0. Similarly doing a QC check on a machine that is not due for service also incurs 0 cost. Any other combination of state of machine and action incurs a cost of -10. This is shown in the following table.

	(S, S)	(S, QC)	(QC, S)	(QC, QC)
(O, O)	-20	-10	-10	0
(O, DS)	-10	-20	0	-10
(DS, O)	-10	0	-20	-10
(DS, DS)	0	-10	-10	-20

2.2 Exact Algorithms

We shall now examine existing algorithms for DEC-POMDPs. Algorithms for solving finite horizon DEC-POMDPs can be classified into three types:

- **Exact Algorithms**. These algorithms find an optimal joint policy.
- **Nash Equilibrium Finding Algorithms**. These algorithms find a Nash Equilibrium of the DEC-POMDP, that is, a locally optimal joint policy. Since an optimal joint policy is also by definition locally optimal, they *may* find an optimal joint policy but do not come with a guarantee of doing so.
- **Approximate Algorithms**. These algorithms find a joint policy that seems to be a 'good' joint policy, but which is not known to be either optimal or locally optimal.

In this section, we shall describe exact algorithms. In the next section, we shall describe algorithms of the other two categories.

As remarked in Chapter 1, there are not many exact algorithms for finite horizon DEC-POMDPs. In fact, only three are known. The three exact algorithms for solving finite horizon DEC-POMDPs are the two backward induction algorithms, Dynamic Programming (henceforth, DP) [HBZ04] and Point Based Dynamic Programming (PBDP) [SC06], and the forward search algorithm Multi Agent A* algorithm (MAA*) [SCZ05]. Some variations of these algorithms also exist (such as a combination of backward induction and forward search [SZ07], the Generalized MAA* algorithm [OSV08] etc.), but the three algorithms represent the various approaches used to tackle this rather difficult problem. The two algorithms, DP and MAA*, are both dynamic programming algorithms but they approach the problem from opposite directions. The DP algorithm is based on the principle of backward induction while MAA* uses forward search. Both algorithms progressively build an optimal T-period joint policy. The third algorithm PBDP is, in comparison with these two algorithms, a somewhat less interesting algorithm because its time requirement is in *all* cases doubly exponential in T - 1.

2.2.1 The DP Algorithm

The DP Algorithm [HBZ04] is a backward induction algorithm, similar in nature to the complete enumeration algorithm [Mon82] used for finite horizon POMDPs. It is based on the fact that the value of a joint policy at a state $\beta \in \Delta(S)$ is a *convex combination* of the values of the joint policy at the corner states of $\Delta(S)$, i.e., members of S. That is, for any $\beta \in \Delta(S)$, the value of a T-period joint policy σ at β can be defined as,

$$V(\beta, \sigma) = \sum_{s \in S} \beta(s) V(s, \sigma) \tag{2.11}$$

$V(s, \sigma)$ is defined recursively as,

$$V(s, \sigma) = V(s, \sigma, \varnothing) \tag{2.12}$$

where the elements of recursion are (2.13), (2.14) and (2.15) given as follows:

For each $s \in S$,

$$V(s, \sigma, \varnothing) = R(s, \sigma(\varnothing)) + \sum_{o \in O} \sum_{s' \in S} \mathbb{P}(s, a, s') \mathbb{G}(a, s', o) V(s', \sigma, o) \tag{2.13}$$

For each state $s \in S$, for each $t = 1$ to T - 2 and for each $\overline{o} \in \overline{O}^t$,

$$V(s, \sigma, \overline{o}) \quad = \quad R(s, \sigma(\overline{o})) + \sum_{o \in O} \sum_{s' \in S} \mathbb{P}(s, \sigma(\overline{o}), s') \mathbb{G}(\sigma(\overline{o}) V(s', \sigma, \overline{o}o) \tag{2.14}$$

$\overline{o}o$ is the tuple of sequences of $(t + 1)$ observations $(\overline{o}_1 o_1, \overline{o}_2 o_2, \ldots, \overline{o}_n o_n)$.

For each $s \in S$, for each $\overline{o} \in \overline{O}^{T-1}$,

$$V(s, \sigma, \overline{o}) \quad = \quad R(s, \sigma(\overline{o})) \tag{2.15}$$

The DP Algorithm iterates for T steps. At each step $t = 1, 2, \ldots, T$, the algorithm first generates, for each agent $i \in I$, the set $\hat{\Pi}_i^t$ obtained by *fully backing up* (described below) the set $\hat{\Pi}_i^{t-1}$, created in the previous step. When $t = 1$, $\hat{\Pi}_i^t$ is just the set A_i. Thereafter, the algorithm prunes each set $\hat{\Pi}_i^t$ by removing from it all t-period policies of agent i that are *very weakly dominated* using the procedure of iterative elimination. At the end of T steps, the algorithms thus creates the set $\hat{\Pi}^T = \times_{i \in I} \hat{\Pi}_i^T$, a subset of $\Pi^T = \times_{i \in I} \Pi_i^T$. The optimal T-period joint policy at α is then obtained by enumerating the members of $\hat{\Pi}^T$.

$$\sigma^* = \arg \max_{\sigma \in \hat{\Pi}^T} \sum_{s \in S} \alpha(s) V(s, \sigma) \tag{2.16}$$

Note that an optimal T-period joint policy at α can also be found by directly enumerating the members of Π^T. But that would require time doubly exponential in T. The objective of the removing very weakly dominated policies at each step is to minimize the size of the set whose members we are required to enumerate at the end of T steps. Therefore the smaller the size of $\hat{\Pi}^T$, the faster we can find an optimal joint policy.

A policy σ_i of agent i is very weakly dominated if, whatever be the initial state of the DEC-POMDP, there exists a joint policy that is optimal at the initial state in which the policy of agent i is not σ_i, but some other policy. The formal definition of a very weakly dominated strategy is as follows. A policy $\sigma_i \in \Pi_i^t$ of agent $i \in I$ is said to be **very weakly dominated** if for every probability distribution γ over the Cartesian set $S \times \Pi_{-i}^t$, there exists another policy $\sigma_i' \in \Pi_i^t$ such that,

$$\sum_{s \in S} \sum_{\sigma_{-i} \in \Pi_{-i}^t} \gamma(s, \sigma_{-i}) \big\{ V(s, (\sigma_i', \sigma_{-i})) - V(s, (\sigma_i, \sigma_{-i})) \big\} \quad \geq \quad 0 \tag{2.17}$$

where $\gamma(s, \sigma_{-i})$ denotes the probability of the pair (s, σ_{-i}) in γ. Note that Π_{-i}^t denotes the set of all possible t-period i-reduced joint policies. An i-reduced joint policy is an $(n$ - $1)$-tuple of policies which does not include the policy of agent i. Whether a given policy of a given agent is weakly dominated or not can be determined by means of a linear program (LP). The following LP determines if a t-step policy σ_i of agent 1 is very weakly dominated or not:

$$\text{Minimize} \quad \epsilon \tag{2.18}$$

Subject To,

$$\sum_{s\in S}\sum_{\sigma_{-i}\in\Pi^t_{-i}} x(s,\sigma_{-i})\big\{V(s,(\sigma'_i,\sigma_{-i}) - V(s,(\sigma_i,\sigma_{-i}))\ \leq\ \epsilon,\quad \forall\sigma'_i\in\Pi^t_i\backslash\{\sigma_i\} \tag{2.19}$$

$$\sum_{s\in S}\sum_{\sigma_{-i}\in\Pi^t_{-i}} x(s,\sigma_{-i})\ =\ 1 \tag{2.20}$$

$$x(s,\sigma_{-i})\ \geq\ 0,\quad \forall s\in S,\ \forall\sigma_{-i}\in\Pi^t_{-i} \tag{2.21}$$

$$\epsilon\ \in\ [-\infty,+\infty] \tag{2.22}$$

If $\epsilon^* \geq 0$, then σ_i is very weakly dominated, where ϵ^* denotes the value of the variable ϵ in an optimal solution to the LP.

The **full backup** of a set $\hat{\Pi}^t_i$ of t-period policies of agent i produces a set $\hat{\Pi}^{t+1}_i$ of $(t+1)$-period policies of the agent. We require the following notation to define the set $\hat{\Pi}^{t+1}_i$. Given an $|O_i|$-tuple $\tilde{\pi} = (\pi^1_i, \pi^2_i, \ldots, \pi^{|O_i|}_i)$, of t-period policies of agent i and an action a of agent i, let $\tilde{\pi}^a$ denote the following $(t+1)$-step policy of agent i (We assume that the observations of agent i are numbered from 1 to $|O_i|$. So given an observation $o \in O_i$, π^o_i is the oth policy in the tuple $\tilde{\pi}$.)

$$\tilde{\pi}^a(\varnothing)\ =\ a \tag{2.23}$$

and for each $o \in O_i$, for each $k = 1$ to t - 1 and for each $\overline{o} \in \overline{O}^k_i$,

$$\tilde{\pi}^a(o\overline{o})\ =\ \pi^o_i(\overline{o}) \tag{2.24}$$

Given a set Z let Perm(Z, m) denote the set of all m-permutations of the members of Z. Then, a full backup of $\hat{\Pi}^t_i$ produces the set $\hat{\Pi}^{t+1}_i$ defined as,

$$\hat{\Pi}^{t+1}_i\ =\ \{\tilde{\pi}^a|\forall a\in A_i, \forall\tilde{\pi}\in \text{Perm}(\hat{\Pi}^t_i, |O_i|)\} \tag{2.25}$$

The main drawback of the DP algorithm is that since it does a full backup at every step, its space requirement increases exponentially (in n and in the number of joint observations) with every step. In practice, the pruning of very weakly dominated policies does make much of a dent in this requirement. As the length of the horizon increases, the DP algorithm quickly starts to run out of space. Of course, once the set $\hat{\Pi}^t$ is created, the set $\hat{\Pi}^{t-1}$ is no longer required and can be cleared from the memory. Thus, in the worst case (that is, no policy is pruned at any step) the algorithm is required to store in memory all possibly T-period joint policies. This means that in the worst case the space required by the algorithm is doubly exponential in T and exponential in n. Additionally, in the worst case, its time requirement is also doubly exponential in T and exponential in n since it enumerates all possible T-period joint policies. Note that it is *not* the case that the DP algorithm can take advantage of the knowledge of the initial state and thereby further reduce the sizes of the sets $\hat{\Pi}^1_i$, $\hat{\Pi}^2_i$, ..., $\hat{\Pi}^T_i$. The DP algorithm simply cannot make use of α. Its principle is such that it can only find an optimal joint policy for α by finding an optimal policy for every state in $\Delta(S)$.

2.2.2 The MAA* Algorithm

The MAA* algorithm [SCZ05] is a forward search algorithm based on the classic A* search algorithm. The major advantage of the MAA* algorithm over the DP algorithm is its minimal space requirement. Unlike the DP algorithm, the MAA* algorithm does not require to store joint policies in memory. Instead, it enumerates joint policies (according to a certain order, described below), computing their values as it goes along, so that at the end of the procedure, a joint policy with the largest value at α is available.

The central idea of the MAA* algorithm (as of the A* algorithm) is to construct an optimal T-period policy incrementally using forward search heuristics. The increments are made as follows. The algorithm first finds a 1-period joint policy such that one of its descendants is a T-period optimal joint policy. Using this 1-period joint policy, the algorithm finds a 2-period joint policy, again such that one of its *descendants* is an optimal T-period optimal joint policy, and so on. Thus, the algorithm iterates for T steps $t = 1, 2, \ldots, T$. At each step t, it generates one t-period joint policy, which we shall call an optimal t-period **parent**, such that one its descendants is a T-period optimal joint policy. Thereby, the joint policy the algorithm finds at the end of step T is an optimal T-period joint policy.

For any $t \leq T$, a T-period policy π' of agent i is said to be a **descendant** of a t-period policy π of the agent if for each integer $k = 1$ to t, for each sequence $\overline{o} \in \overline{O}_i^{k-1}$ of k observations, $\pi(\overline{o}) = \pi'(\overline{o})$. Similarly, a $(t + 1)$-period policy π' of agent i is said to be a **child** of a t-period policy π of the agent if for each integer $k = 1$ to t, for each sequence $\overline{o} \in \overline{O}_i^{k-1}$ of k observations, $\pi(\overline{o}) = \pi'(\overline{o})$. In order to identify an optimal t-period parent, the potential value of every child of the optimal $(t - 1)$-period parent is computed. The child with the largest potential value is retained as the optimal t-period parent. The potential value computation of a t-period joint policy is the key computational step of the algorithm. The potential value of a t-period joint policy is an upper bound on the value of every descendant of the joint policy. Therefore, the definition of the potential value $PV(\alpha, \sigma)$ of a t-period joint policy σ consists of two parts:

$$PV(\alpha, \sigma) \quad = \quad V(\alpha, \sigma) + H^{T-t}(\alpha, \sigma) \tag{2.26}$$

$V(\alpha, \sigma)$ is the value of the joint policy and $H^{T-t}(\alpha, \sigma)$ is the upper bound on the value (expected reward) achieved by any descendant for the remaining $T - t$ periods.

[SCZ05] propose three heuristics for calculating $H^{T-t}(\alpha, \sigma)$: the MDP heuristic, the POMDP heuristic and the DEC-POMDP heuristic. The MDP and POMDP heuristic are motivated by the fact that the value of an optimal $(T - t)$-period DEC-POMDP policy at any state $\beta \in \Delta(S)$ is bounded by above by the values of the optimal $(T - t)$-period MDP and the optimal $(T - t)$-period POMDP policies at β.

The value of an optimal t-period MDP policy at β, denoted by $V_M^t(\beta)$, is defined recursively as,

$$V_M^t(\beta) \quad = \quad \sum_{s \in S} \beta(s) V_M^t(s) \tag{2.27}$$

where for each $s \in S$ and for each t,

$$V_M^t(s) = \begin{cases} \max_{a \in A} \left\{ R(s, a) + \sum_{s' \in S} \mathbb{P}(s, a, s') V_M^{t-1}(s') \right\}, & \text{if } t > 1 \\ \max_{a \in A} R(s, a), & \text{otherwise} \end{cases} \tag{2.28}$$

Similarly, the value of an optimal t-period POMDP policy at β, denoted by $V_P^t(\beta)$, is defined recursively as,

$$V_P^t(\beta) = \sum_{s \in S} \beta(s) V_P^t(s) \tag{2.29}$$

where for each $s \in S$ for each t,

$$V_P^t(s) = \begin{cases} \max_{a \in A} \left\{ R(s,a) + \sum_{s' \in S} \sum_{o \in O} \mathbb{P}(s,a,s') \mathbb{G}(a,s',o) V_P^{t-1}(s') \right\}, & \text{if } t > 1 \\ \max_{a \in A} R(s,a), \quad \text{otherwise} \end{cases} \tag{2.30}$$

Given a t-period optimal DEC-POMDP policy σ^*, there holds,

$$V(\alpha, \sigma^*) \leq V_P^t(\alpha) \leq V_M^t(\alpha) \tag{2.31}$$

The MDP and the POMDP heuristics are employed as follows. Let Prob.$(s|\alpha, \sigma)$ denote the probability that the state of the process is s in period $t + 1$ if the joint policy σ was executed in the initial state α. Then,

$$H^{T-t}(\alpha, \sigma) = \sum_{s \in S} \text{Prob.}(s|\alpha, \sigma) V_H^{T-t}(s) \tag{2.32}$$

where H denotes the heuristic employed. So, if the MDP heuristic is used,

$$V_H^{T-t}(s) = V_M^{T-t}(s) \tag{2.33}$$

Similarly, if the POMDP heuristic is employed,

$$V_H^{T-t}(s) = V_P^{T-t}(s) \tag{2.34}$$

Finally, if the DEC-POMDP heuristic is used,

$$V_H^{T-t}(s) = V(s, \sigma^*) \tag{2.35}$$

where σ^* is an optimal (T - t)-period joint policy for s, found according to some algorithm, conceivably MAA* itself.

Thus, the MAA* algorithm relies on the rapidity of the computation of the quantity $V_H^{T-t}(s)$. The MDP heuristic is fast to compute, but it serves as a loose upper bound on the value of the descendants of a t-period joint policy. The POMDP heuristic is tighter but requires more time. The DEC-POMDP heuristic is even better, but takes even longer to compute. The tightness of a heuristic plays an important role in the runtime of the algorithm for the following reason. If the value of a child of an optimal (T-1)-period parent equals the potential value of the parent itself, then clearly the search for an optimal T-period joint policy can be terminated; the child is such a joint policy. In most cases, the MDP and the POMDP heuristic are not tight enough to allow us to make this discovery without enumerating all the children of the optimal (T - 1)-period parent. The number of children of this parent is of course doubly exponential in the horizon.

While overall the MAA* algorithm could be said to be an improvement over the DP algorithm, the fact is that the low space requirement of the algorithm is offset by its high time requirement. in the worst case, the time requirement of the algorithm is doubly exponential in T. Hence, the algorithm quickly runs out of time as the horizon increases. Of course, the time requirement of the DP algorithm is as bad.

2.2.3 Point Based Dynamic Programming (PBDP)

The PBDP algorithm [SC06] is a backward induction dynamic programming algorithm that determines an optimal joint policy for the given initial state α. It is similar to Wilson's dynamic programming algorithm [Wil72]. A schematic description of the PBDP algorithm is as follows. The algorithm first generates the set of states in $\Delta(S)$ that are reachable (realizable) at the end of T periods by all possible T-period joint policies. From this set the algorithm identifies those states are reachable by the last joint actions of an optimal joint policy. In doing so, the algorithm determines the last joint actions of an optimal joint policy. It thus builds the last part of an optimal joint policy. Thereafter, the algorithm determines the states that are reachable at the end of T - 1 periods by those T-period joint policies whose last actions are as determined by the algorithm. Thus, the algorithm determines the set of last two joint actions to be taken by an optimal joint policy. The algorithm continues building an optimal joint policy in this manner till at the end of T iterations, it has built an optimal joint policy.

Whereas the DP algorithm has a very high (doubly exponential) worst case space and time requirement and the MAA* algorithm has a very high (doubly exponential) worst case time requirement, the PBDP algorithm has a very high (doubly exponential) *every case* time requirement. This is because in order to find the last joint actions of an optimal joint policy, the very first step of the algorithm consists of enumerating all (T - 1)-period joint policies. Since the number of (T - 1)-period joint policies is doubly exponential in (T - 1), the algorithm essentially faces its biggest bottleneck at the very first step. This makes the PBDP algorithm somewhat impractical.

2.3 Inexact Algorithms

The exact algorithms described in the previous section run out of space and or time as the horizon of the problem increases or as the number of agents increase. In fact, the algorithms are practical only for 2-agent finite horizon DEC-POMDPs and for very small horizons (< 4 or 5). Nash Equilibrium Finding algorithms have therefore attracted much attention. The principle algorithms of this type are the Coevolution Algorithm [CSC02], the joint equilibrium-based search for policies (JESP) algorithm [NTY$^+$03] and the Continuous-Space JESP Algorithm [VNTY06].

A Nash equilibrium is a locally optimal joint policy and potentially, such a joint policy can be found using lesser resources than an optimal joint policy. There are two caveats in this regard. First, as proved in [KM92], finding a Nash equilibrium is also an NP-hard problem in the size of the DEC-POMDP (defined as an exponent of the horizon; alternatively, it can be described as being NEXP-hard if the size of the DEC-POMDP is defined in terms of its horizon). Second, a Nash equilibrium can be arbitrarily sub-optimal since while every optimal joint policy is necessarily a Nash Equilibrium, not every Nash Equilibrium is an optimal joint policy. In other words, the difference between the value of an optimal joint policy and a Nash equilibrium can be very large.

A joint policy that constitutes a Nash equilibrium is defined as follows. A T-period joint policy σ is a **Nash Equilibrium** if there holds,

$$V(\alpha, \sigma) \quad \geq \quad V(\alpha, (\pi, \sigma_{-i})), \quad \forall i \in I, \forall \pi \in \Pi_i^T \tag{2.36}$$

Thus a joint policy is locally optimal if any joint policy obtained on replacing the policy of only *one* agent in the joint policy does not have a value that is larger than the value of the unchanged joint policy. To see that the difference between the values of an optimal joint policy and a Nash equilibrium can be very large, consider the following matrix:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 3 \\ 0 & 3 & 4 \end{pmatrix} \tag{2.37}$$

The entries of the matrix represent values of joint policies for a 2-agent DEC-POMDP. Each agent has three policies, 1, 2, and 3. Agent 1's policies index the rows and agent 2's policies index the columns. The joint policy (2, 2) is not locally optimal because agent 1 can change his policy from 2 to 3, and the agents can thereby obtain an expected reward of 3. Neither (3, 2) nor (2, 3) are locally optimal. Now, the joint policy (1, 1) is locally optimal but it has a an expected reward of 1, which is lesser than the expected reward of (2, 2). Finally, the optimal joint policy is (3, 3) which is (by definition) also locally optimal and has an expected reward of 4. By multiplying each expected reward by a very large number, the difference between the expected reward of a joint policy that is not locally optimal (here (2, 2)) and one that is (here (1, 1)) can be made arbitrarily large. The above example can be considered to be a typical case for a DEC-POMDP. The number of joint policies that are locally optimal but are not optimal is large.

Despite the drawbacks of a Nash Equilibrium as a solution concept for DEC-POMDPs, two things can be said in its favor. Algorithms that are designed to find a Nash equilibrium, in practice often find an optimal joint policy. Second, finding a Nash equilibrium of a DEC-POMDP in practice does exhibit worst case requirement of space and/or time.

The JESP algorithm (and its variant the dynamic programming JESP Algorithm) is based on the following principle. If the policies of all except one agents are fixed, the problem essentially reduces to a POMDP. A solution of this POMDP is a policy that is a best response to the fixed policies of the remaining agents. The resulting joint policy is a joint policy in which at least one agent's policy is a best response to the reduced joint policy formed by the remaining agents' policies. In a Nash equilibrium, each agent's policy is a best response to the reduced joint policy formed by the remaining agents' policies. Now, if we iteratively fix the policies of n - 1 agents, and solve for the resulting POMDP problem, at the end of each iteration, one of the two things would happen: either we would obtain a joint policy with the same value as the the joint policy we obtained in the previous iteration or we would obtain a joint policy with a larger value than the one obtained in the previous iteration. We would never obtain a joint policy with a smaller value. So that a point would arrive where we would obtained a joint policy whose value cannot be improved upon by changing the policy of just one agent. Thus, in this joint policy, we are assured that each policy is a best response to the reduced joint policy formed by the remaining policies. Using this principle, the JESP algorithm find a Nash equilibrium of a DEC-POMDP. The Coevolution Algorithm [CSC02] is based on a similar principle.

The Continuous-Space JESP (CS-JESP) Algorithm extends the JESP Algorithm in a significant way by finding the smallest set of joint policies such that for each joint policy in the set there exists at least one state $\beta \in \Delta(S)$ such that if β is the initial state of the DEC-POMDP, the joint policy is a Nash equilibrium. The CS-JESP Algorithm is thus the analogue of the DP Algorithm for finding Nash Equilibria of the DEC-POMDP.

2.3.1 Approximate Algorithms

Existing approximate algorithms find a joint policy that seems to be a good joint policy but for which the difference between its value and the value of the optimal joint policy is not known. Two algorithms of this category are the Approximate PBDP Algorithm [SC06] and the memory-bounded dynamic programming (MBDP) algorithm [SZ07].

The approximate PBDP algorithm is a variant of the PBDP algorithm in which instead of generating all possible reachable states, only a fixed number of states are generated. The contention of the algorithm is that if these states are carefully chosen and well spread out in the state simplex, joint actions assigned for those states as optimal would also turn out to be optimal for states not considered. By limiting the number of reachable states considered, the algorithm essentially avoids a complete enumeration of joint policies as is required in the PBDP algorithm. However, [SC06] provide only a very loose bound on the loss in value incurred by not considering all reachable states.

The MBDP algorithm, as its name suggests, finds a joint policy that is optimal for a given amount of resources (space). In other words, if the algorithm is given more space, it may find a joint policy with a larger value. The MBDP algorithm consists of two separate ideas. The first idea consists of combining the backward-induction and the forward-search aspects of respectively the DP Algorithm and the MAA* Algorithm. So, after a set of t-period policies is generated for all the agents using backward induction, forward-induction heuristics are used to identify those policies from this set that are potentially optimal given the initial state of the problem. Those policies that are not provably not optimal are removed from the set.

The second idea consists of fixing the number of useful trees are that allowed to be in the set of useful policies. That is, once useful policies are identified using forward-induction heuristics, only some of these useful policies are retained in the set. The other useful policies are removed from the set. The number of policies retained in each set at each step is a constant. This reduced set of useful t-step policies is then subject to a full-backup to produce a set of $(t + 1)$-step policies. Thus, at each step of the algorithm a constant number of policies are created. Thereby, the exponential growth in the number of policy trees added at each step is avoided. However, [SZ07] do not provide bounds for the loss in value incurred as a result of discarding possibly useful trees.

2.4 Computational Experience

The following tables give an idea of the capacity of the DEC-POMDP algorithms described above. The tables show the time taken by the algorithms to solve a given DEC-POMDP where 'solve' means finding an optimal joint policy or a Nash Equilibrium or just a 'good' joint policy, depending on the algorithm. The figures are taken from the papers in which the algorithms were presented.

The following table shows the performance of the algorithms the MA-Tiger problem [NTY$^+$03]. This problem is a 2-agent DEC-POMDP with 2 states and with 3 actions per agent and 2 observations per agent.

Algorithm	Horizon	Time Taken	Horizon	Time Taken
MAA*	3	4 s	4	$>$ 1 month
Recursive MAA*	3	4 s	4	2.5 h
JESP	3	317 s	4	-
DP-JESP	3	0	4	0.02 s
MBDP	3	0.19 s	4	0.46 s

'-' indicates a time out (although it is not reported, it is safe to assume that the time out is probably of a few hours). Note that Recursive MAA* is MAA* with the DEC-POMDP heuristic computed using MAA* itself.

The following table shows the performance of the algorithms on a version of the MABC problem [HBZ04]. This problem is a 2-agent DEC-POMDP with 4 states and with 2 actions per agent and 2 observations per agent.

Algorithm	Horizon	Time Taken	Horizon	Time Taken	Horizon	Time Taken
DP	3	5 s	4	900 s	5	-
MAA*	3	$<$ 1 s	4	3 h	5	-
Recursive MAA*	3	$<$ 1 s	4	1.5 h	5	-
PBDP	3	$<$ 1 s	4	2 s	5	$>$ 30 h
Approx. PBDP	3	$<$ 1 s	4	$<$ 1 s	5	10 s
MBDP	3	0.01	4	0.01	5	0.02 s

'-' indicates that the algorithm either ran out of time or space.

As these figures show, exact algorithms are grossly incapable of tackling even the smallest of DEC-POMDPs. This is not surprising in view of the negative complexity result for DEC-POMDPs. Therefore, the question is not so much whether these algorithms can be improved upon in the absolute, but rather if a relative improvement can be achieved. In other words, can we push the computational envelop a bit further on this problem? This is one of the questions addressed in our thesis. There is of course a more relevant reason for pushing this envelop. As described, exact algorithms serve as a basis for approximate algorithms, and as the figures show, approximate algorithms are seemingly fast, even though it is difficult to know the quality of the joint policy returned by these algorithms (in other words, we do not yet have a clear idea about the trade-off of time versus quality made by these algorithms). So, a more efficient exact algorithm is important from this perspective as well.

2.5 Mathematical Programming Basics

In this section, we briefly define some of the mathematical programming formulations we shall use in this thesis. More details and theory about these formulations can be obtained from any standard text on constrained and unconstrained optimization. We have consulted the texts [PS82], [Lue84] and [Fle87] for our work. The discussion in this section is rudimentary and is intended only for those who are not acquainted at all with mathematical programming.

We begin by describing a mathematical program (MP). An MP is a mathematical description of a problem of optimization. We have already seen some MPs in Chapter 1. An MP consists of three elements:

1. A set V of *variables*.
2. A set C of *constraints*.
3. A function f from V to $\mathbb{R}$ which is required to be optimized (minimized or maximized).

A constraint is an equation or an inequality defined over the set V. An MP need contain contain all three elements. In some MPs, C is not defined. Such MPs are said to define problems of *unconstrained* optimization. MPs for which C is defined are said to define problems of *constrained* optimization. Similarly, in some MPs, f is not defined. Thus, MPs can be of three forms: (V, C, f), (V, C) and (V, f).

An **optimal solution** to an MP (V, C, f) is an assignment of real numbers to the variables in V such that all the constraints in C are satisfied and f is optimized. An optimal solution to an MP (V, C) is an assignment of real numbers to the variables in V such that all the constraints in C are satisfied. Finally, an optimal solution to an MP (V, f) is an assignment of real numbers to the variables in V that optimizes f. Note that not every MP has a solution. In three cases, an MP cannot not have a solution:

- Two or more constraints contradict one another.
- f is unbounded from above, and we are maximizing f.
- f is unbounded from below, and we are minimizing f.

For MPs of the type (V, C, f), an assignment of values to the variables in V which satisfies all the constraints in C but does not necessarily optimize f is called a **feasible solution**. For MPs of the type (V, C), every feasible solution is also an optimal solution.

Our thesis deals with problems of constrained optimization. Hence, the MPs we shall use are of the type (V, C, f) or (V, C). We shall use mainly four types of mathematical programs in the thesis:

- Nonlinear program (NLP).
- Linear Program (LP).
- 0-1 Integer Linear Program (0-1 ILP).
- 0-1 Mixed Integer Linear Program (0-1 MILP).

All programs shall involve the maximization of f.

In an NLP, either f is a polynomial function of degree 2 or more, and/or one or more constraints are polynomials of degree 2 or more. We have already seen an example of an NLP in Chapter 1, the one that defines a 2-agent TDP, and generally, decentralized problems are defined as NLPs. In an LP, f is a polynomial of degree 1 and every constraint in C is also of a polynomial of degree 1. In an NLP or an LP, each variable is a **continuous variable**. An continuous variable is one that is not restricted to take non-integer values in any solution to the LP or NLP.

A 0-1 ILP is an LP in which each variable is a **boolean variable** or a **0-1 variable**. A 0-1 variable is restricted to take a value of either 0 or 1 in any solution to the ILP. Finally, a 0-1 MILP is an LP in which some variables are 0-1 variables and the remaining ones are continuous variables. NLPs can be oftentimes rewritten as 0-1 ILPs or 0-1 MILPs. Therefore, decentralized problems can be expressed as NLPs or 0-1 ILPs/MILPs. Indeed, the finite horizon DEC-POMDP problem can be expressed as an NLP or (as we show in Chapters 4 and 5) as a 0-1 ILP/MILP.

Our convention for LPs shall be that variables shall be represented in small letters at the end of the alphabet such as x, y, z, ..., etc. Constants or the data of the LP shall be usually represented by small letters at the beginning of the alphabet such as a, b, c, ..., etc. An optimal solution to an LP shall be denoted be placing an asterisk on the variable. Thus, if an MP uses a variable x, the value of x in an optimal solution shall be denoted by x^*.

NLPs have two types of solutions, a locally maximum solution (which is a local maximum point of f) and a globally maximum solution (which is a global maximum point of f). If f is nonconcave (which is it, in most NLPs), there exist no generalized methods that guarantee finding a globally optimal solution to an NLP; existing methods only guarantee finding a locally maximum solution. However, the problem of finding even a locally maximum solution to an NLP is NP-hard.

In an LP, 0-1 ILP or a 0-1 MILP every locally maximum solution is also a globally maximum solution. The problem of finding an optimal solution to a 0-1 ILP or a 0-1 MILP is NP-hard [Pap80]. On the other hand, finding an optimal solution to an LP is a problem of polynomial complexity [Meg87]. The most important algorithm for solving an LP is the simplex algorithm. (Mixed) integer linear programs can be solved by the *branch and bound* (BB), which solves a tree of LPs. Each LP in the tree is solved through the simplex algorithm. The BB method is described in detail in Chapter 4 (Section 4.4.2).

2.6 Summary

In this chapter we have described the DEC-POMDP model and shown how it is used to formulate practical decentralized problems. We have analyzed existing exact and inexact algorithms for the problem of finding an optimal joint policy for a problem modeled as a DEC-POMDP. We have described in brief the computational experience of these algorithms. In the next chapter, we shall present the sequence-form of a policy, a representation of a finite horizon policy that shall enable us to conceive faster algorithms for finding an optimal finite horizon joint policy for a DEC-POMDP.

Chapter 3

The Sequence Form Of A Policy

3.1 Introduction

In this chapter, we present a new representation of a finite horizon policy of an agent in a DEC-POMDP. This representation is called a *policy in the sequence-form*. It was introduced by D. Koller, B. von Stengel and N. Megiddo through a series of papers - [KMvS94], [KM96], [vS96] - on the subject of solving extensive-form games. The representation of a policy presented in Chapter 2 (Section 2.1, page 68) is called the *canonical form*.

A policy in the sequence-form of an agent is a conditional probability distribution over the set of *histories* of the agent. A history is a sequence of actions and observations. An agent takes actions according to the probabilities defined by this distribution. Those histories that receive a nonzero probability in this conditional probability distribution may or may not have a nonzero probability of occurring when the policy is executed; those that receive zero probability will definitely not occur. A policy in the sequence-form can be found by solving a set of linear equations. This set contains approximately one linear equation per history. The main insight regarding the sequence-form of a policy is that whereas the number of T-period policies in the canonical form of an agent is doubly exponential in T, the number of histories produced collectively by the policies is only exponential in T. Hence, the space required to find a policy in the sequence-form is only exponential in T.

The implication of using the sequence-form of the policy is therefore that finding an optimal joint policy in the sequence-form requires much lesser time and space than existing algorithms require to find an optimal joint policy in the canonical form. The algorithms that find an optimal joint policy in the canonical form require time and/or space that is doubly exponential in the horizon to find an optimal joint policy, as described in Chapter 2. Finding an optimal joint policy in the sequence form requires space that is exponential in the horizon, and in practice, requires lesser (by a magnitude or an order or two) time.

Since a policy in the sequence-form of a policy is a conditional probability distribution over the set of histories of an agent, in finding an optimal joint policy in the sequence form, we resort to a mathematical programming approach rather than a dynamic programming approach. Our approach is now to *combine* histories of agents into an optimal joint policy, instead of building an optimal joint policy incrementally. Note that a policy in the sequence-form is just a different representation of a policy. Every policy in the sequence-form can be converted into an equivalent

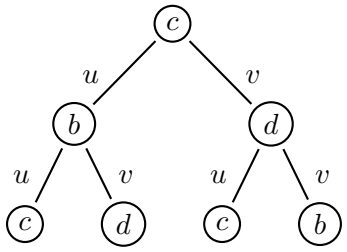


Figure 3.1: A 3-Period Policy In The Tree Form.

pure or mixed policy in the canonical form where equivalence is in terms of the probability of taking an action for a given sequence of observations.

3.2 Informal Description

We shall take up a few examples in this section to describe a policy in the sequence-form. As stated, a policy in the sequence-form of an agent is a conditional probability distribution over the set of the agent's histories. These probabilities can be stored in a vector containing one entry per history, the entry representing the history's conditional probability. The conditional probability of a history in a policy shall be called its *weight*. The weight of history h in policy p shall be denoted by $p(h)$.

A finite horizon policy in the canonical form can be represented as a rooted tree. Consider the 3-period canonical policy shown in Figure 3.1. When using this, the agent takes action c in period 1 with probability 1. In period 2, if he receives observation u, then he takes action b with probability 1. So, we can say that, in using this policy, the agent takes action c in period 1 and action b in period 2 with probability 1 *if* he receives observation u in period 2. This probability is not a function of the DEC-POMDP model or of the policies of the other agents. It is only a conditional probability, conditional on the agent receiving u at period 2. The actual probability that the agent takes actions c and b in periods 1 and 2 respectively and receives observation u in period 2 certainly depends on the DEC-POMDP model and the policies of the other agents. Thus, we can say that, in using this policy, the conditional probability of the history cub is 1. Similarly, in using this policy, the probability that the agent takes action c in period 1, action d in period 2 and action c in period 3 is 1 *if* he receives observation v in period 2 and observation u in period 3. Thereby, the conditional probability of the history $cvdub$ is 1. Conditional probabilities of some histories are 0 under this policy. For instance, the conditional probability of the history $cubvc$ is 0 because if the sequence of observations received by the agent till period 3 is uv, then according to the policy, the agent takes actions c, b and d respectively in periods 1, 2 and 3, and not actions c, b and c respectively in periods 1, 2 and 3.

The sequence form of the policy given in Figure 3.1 is shown in Figure 3.2. It is a vector containing these conditional probabilities or weights as shown below. Only those histories which receive a nonzero weight are shown in the vector.

In a pure policy in the sequence form, each entry in the policy is either a 0 or 1. An agent uses a pure policy p in the sequence-form as follows. At each period, the agent takes an action

History	Weight	History	Weight
c	1	cub	1
cvd	1	$cubuc$	1
$cubvd$	1	$cvduc$	1
$cvdvb$	1		

Figure 3.2: A 3-Period Policy In The Sequence Form.

History	Weight	History	Weight
b	1	bub	1
bvd	1	$bubuc$	1
$bubvd$	1	$bvdub$	1
$bvdvc$	1	$bubucud$	1
$bubucvb$	1	$bubvdub$	1
$bubvdvd$	1	$bvdubuc$	1
$bvdubvc$	1	$bvdvcud$	1
$bvdvcvc$	1		

Figure 3.3: A 4-Period Policy In The Sequence Form.

as a function of the observation he receives in the period, and the history of actions taken and observations received till the previous period. If h_t is the history that has occurred till period t and if o is the observation received at period $t+1$, then at period $t+1$, the agent takes that action a for which $p(h_t oa) = 1$; there will be only one such action.

Let the sequence-form policy shown in Figure 3.2 be denoted by p_1. In following this policy, the agent takes action c at period 1. Then, if the observation he has received in period 2 is v, he takes action d at period 2 because $p_1(cvd) = 1$. He does not take action b or c at this period because $p_1(cvb) = 0$ and $p_1(cvc) = 0$. Similarly, if at the period 3, the observation received is u and the history that has occurred till the end of period 2 is cvd, then he takes action c because $p_1(cvduc) = 1$; he does not take action b or d at this period because $p_1(cvdub) = 0$ and $p_1(cvdud) = 0$.

Figure 3.3 gives another example of a policy in the sequence-form. Again, the actions of the agent are b, c and d, and the observations are u and v. The longest history in this policy contains 4 actions. So this is a 4-period policy. It is assumed that the weight of any history conceivable from these actions and observations that does not appear in the table is 0. So, for instance, the history $budvc$ does not appear in the policy, implying its weight is 0. Let this policy be denoted by p_2. In using this policy, the agent takes action b in period 1 because $p_2(b) = 1$. In period 2, if he receives observation u, he takes action b because $p_2(bub) = 1$ and takes action d if receives observation v because $p_2(bvd) = 1$. In period 3, if observation u is received and if the history of actions taken and observations received prior to period 3 is bvd, then the agent takes action b since $p_2(bvdub) = 1$. Similarly, in period 4, if observation v is received and the history occurred till that step is $bubvd$, then the agent takes action d since $p_2(bubvdvd) = 1$. Assume that in period 4, the history occurred till that period is $bucub$, then it can only mean that the agent has not been following the policy till period 3 because $p_2(bucub) = 0$ according to this policy.

History	Weight	History	Weight
b	0.6	*c*	0.4
bub	0.6	*cub*	0.4
bvc	0.6	*cvc*	0.4
bubub	0.12	*bubuc*	0.48
cubub	0.08	*cubuc*	0.32
bubvc	0.6	*cubvc*	0.4
bvcub	0.6	*cvcub*	0.4
bvcvb	0.42	*bvcvc*	0.18
cvcvb	0.28	*cvcvc*	0.12

Figure 3.4: A 4-Period Stochastic Policy In The Sequence Form.

The weights in a policy in the sequence-form need not be restricted to 0 or 1. They can assume values in the interval [0, 1] leading to a *stochastic* policy in the sequence-form. Thereby, a policy in the sequence-form in which the weights are 0 or 1 shall be called a *pure* policy. The previous two examples were examples of pure policies. Figure 3.4 gives an example of 4-period stochastic policy in the sequence-form. Again, only histories with nonzero weights in the policy are shown.

An agent uses a stochastic policy in the sequence form in the same manner as he uses a pure policy in the sequence-form with one difference. In a stochastic policy - be it in the canonical form or the sequence-form - in any period, an agent may choose an action probabilistically. Therefore, when using a stochastic policy p in the sequence-form, in period $t+1$, upon receiving observation o, the agent chooses each action a with probability $p(h_toa)/p(h_t)$. Unlike the case of a pure policy in the sequence-form, here the denominator $p(h_t)$ need not be always 1.

Let the policy given in Figure 3.4 be denoted by p_3. In using p_3, in period 1, the agent takes action b with probability 0.6 because $p_3(b) = 0.6$ and action c with probability 0.4 because $p_3(c) = 0.4$. In period 2, if he receives observation u and if he has taken action b in period 1, he takes action b with probability 1 because $p_3(bub)/p_3(b) = 0.6/0.6 = 1$. Similarly, in period 2, if he receives observation u and if he has taken action c in period 1, he takes action b with probability 1 because $p_3(cub)/p_3(c) = 0.4/0.4 = 1$. This means, that in period 2, regardless of the action taken in period 1, if he receives observation u, he takes action b. To take one more example, if in period 3, the observation received is v and the history that has occurred till the end of period 2 is cvc, then he takes action b with probability $p_3(cvcvb)/p_3(cvc) = 0.28/0.4 = 0.7$, and he takes action c with probability $p_3(cvcvc)/p_3(cvc) = 0.12/0.4 = 0.3$.

3.3 Formal Definition

We shall now define a policy in the sequence-form in a formal manner. We begin by defining a history.

We define a **history** of agent $i \in I$ to be a sequence of odd length in which the elements in odd positions are actions of the agent (members of A_i) and those in even positions are observations of the agent (members of O_i). A history thus has one more action than it has observations. We

define the **length** of a history to be the number of actions in the history. A history of length 1 just an action; it does not have any observations. A history of length T shall be called a **terminal history**. Histories of lengths lesser than T shall be called **nonterminal** histories.

We shall denote by $\mathcal{H}_i^t$ the set of all possible histories of length t of agent i, $t \geq 1$, conceivable from the sets A_i and O_i. Thus, $\mathcal{H}_i^1$ is just the set of actions A_i. We shall denote by $\mathcal{H}_i$ the set of histories of agent i of lengths less than or equal to t. That is $\mathcal{H}_i = \mathcal{H}_i^1 \cup \mathcal{H}_i^2 \cup \ldots \cup \mathcal{H}_i^t$. The set $\mathcal{H}_i^T$ of terminal histories of agent i shall also be denoted by $\mathcal{E}_i$. The set $\mathcal{H}_i \backslash \mathcal{H}_i^T$ of nonterminal histories of agent i shall be denoted by $\mathcal{N}_i$. Thus, $\mathcal{H}_i = \mathcal{N}_i \cup \mathcal{E}_i$.

We define a **joint history** of length t to be an n-tuple $j = (j_1, j_2, \ldots, j_n)$ where for each $i \in I$, j_i is a history of length t of agent i. Alternatively (and equivalently), we define a joint history of length t to be a sequence of length $2t$ - 1 in which elements in even positions are joint actions and those in odd positions are joint observations. A joint history of length 1 is just a joint action. A joint history of length T shall be called a *terminal* joint history. Joint histories of lengths less than T shall be called *nonterminal* joint histories. Given a joint history j, the history of agent i in it shall be denoted by j_i. The set $\times_{i \in I} \mathcal{H}_i^t$ of joint histories of length t shall be denoted by $\mathcal{H}^t$. The set $\mathcal{H}^T$ of terminal joint histories shall also be denoted by $\mathcal{E}$.

An **i-reduced joint history** j' of length t is an (n-1)-tuple of histories of length t, one history in the tuple per agent in I except for agent i. The set of i-reduced joint histories of length t shall be denoted by $\mathcal{H}_{-i}$. The set of i-reduced terminal joint histories shall be denoted by $\mathcal{E}_{-i}$. Given an i-reduced joint history j' of length t and a history h of length t of agent i, (h, j') shall denote the joint history of length t in which the histories of all agents except i are according to j' and the history of agent i is h.

As described in the previous section, a policy in the sequence-form of agent i assigns every possible history in $\mathcal{H}_i$ a conditional probability. If the policy is pure, it assigns either a 0 or 1 to each history in $\mathcal{H}_i$. If it is stochastic, it assigns a number in [0, 1]. We define a **T-period policy in the sequence-form** of agent i to be a function p from $\mathcal{H}_i$ to [0, 1] such that,

$$\sum_{a \in A_i} p(a) = 1 \tag{3.1}$$

$$-p(h) + \sum_{a \in A_i} p(hoa) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{3.2}$$

where hoa denotes the history obtained on concatenating o and a to h. This definition appears in a slightly different form as Lemma 5.1 in [KMvS94]. The number $p(h)$ is called the **weight** of h in p. In a **pure** policy in the sequence-form, the weight of each history is either 0 or 1. In a **stochastic** policy in the sequence-form, the weight of each history can be any number in the interval [0, 1]. We define the **support** of a policy to be the set of histories that receive a nonzero weight in it. The support $S(p)$ of a policy of agent i is the set $\{h \in \mathcal{H}_i | p(h) > 0\}$.

To reiterate from the previous section, agent i uses a policy p in the sequence-form as follows. At the firs step, the agent takes $a \in A_i$ with probability $p(a)$. Let h_t denote the history formed by the actions taken by the agent and observations received by him till step t, $t \geq 1$. If at the $(t + 1)$th step, the agent receives observation o, then he takes action $a \in A_i$ with probability $p(h_t oa)/p(h_t)$, where $h_t oa$ denotes the history obtained by concatenating o and a to h_t.

The weight $p(h)$ of a history $h = a^1, o^1, a^2, o^2, \ldots, a^t$ of length t of agent i in a policy p of agent i in the sequence-form is the probability of the agent taking the sequence of actions $a^1, a^2, \ldots, a^t$ till period t *if* the sequence of observations he has received period t is $o^1, o^2, \ldots, o^{t-1}$ *and* the agent has taken actions till period t according to p (as described above). That is,

$$p(h) \quad = \quad \text{Prob.}(a^1, a^2, \ldots, a^t | o^1, o^2, \ldots, o^{t-1}) \tag{3.3}$$

The weight of a history in a policy is *not* the probability of the history occurring. The weight is merely an expression of the agent's proclivity or desire that the history occur. Thus, it is possible that $p(h) > 0$ for some history h and policy p, but h has a probability of 0 of occurring when the policy is executed. As an example, consider a history *buc*. Say s is the initial state of the problem. If the joint observation function $\mathbb{G}$ is such that $\mathbb{G}((b, a_{-i}), s, (u, o_{-i})) = 0$ for every i-reduced joint action a_{-i} and for every i-reduced joint observation o_{-i}, then even if $p(buc) = 1$, *buc* will never occur. So if $p(h) > 0$, then it is possible that the history occurs when the agent executes p with a certain probability, which may even be 0. On the other hand, if $p(h) = 0$, then h will certainly not occur when p is executed; the agent himself is ruling out that possibility.

The set of policies in the sequence-form of agent i shall be denoted by X_i. The set of pure policies in the sequence-form shall be denoted by $\hat{X}_i \subset X_i$. The size of $\hat{X}_i$ and the size of Π_i^T - the set of pure policies in the canonical form *is the same*. They are both doubly exponential in T. The following lemmas show this.

Lemma 3.1. *For $t \leq T$, the number of histories of length t that are in the support of a pure policy in the sequence-form of agent i is $|O_i|^{t-1}$.*

Proof: In a pure policy, the weight of each history is either 0 or 1. Those that are in its support have a weight of 1. Due to (3.1), the number of histories of length 1 (i.e., actions) that are in the support of a pure policy of agent i is 1. Thereby, due to (3.2), the number of histories of length 2 that are are in the support of a pure policy is $1 \times |O_i|$. Continuing, the number of histories of length 3 that are in the support of a pure policy is $1 \times |O_i| \times |O_i| = |O_i|^2$, and so on. Another way to arrive at this figure is as follows. A history of length t has t actions and t - 1 observations. The number of possible sequences of t - 1 observations that can be conceived from the set O_i is $|O_i|^{t-1}$. For each of these sequences, the support of a pure policy of the agent must contain one sequence of k actions. Hence, the number of histories of length k in the support of a pure policy of agent i is $|O_i|^{t-1}$. *Q.E.D*

Lemma 3.2. *The number of pure policies in the sequence form of agent i is $|A_i|^{\frac{|O_i|^T - 1}{|O_i| - 1}}$.*

Proof: According to Lemma (3.1), the support of a pure policy of agent i contains $|O_i|^0$ histories of length 1, $|O_i|^1$ histories of length 2, $|O_i|^2$ histories of length 3 etc. Now, the number of ways we can select the last action of a history is obviously $|A_i|$. Hence, the number of ways in which we can select histories of length t to be in the support of a pure policy of agent i is $|A_i|^{|O_i|^{t-1}}$. Hence, the number of ways we can select histories of length 1, of length 2, of length 3, ..., of length T is, $\prod_{t=1}^{T} |A_i|^{|O_i|^{t-1}} = |A_i|^{\frac{|O_i|^T - 1}{|O_i| - 1}}$ since $\sum_{t=1}^{T} |O_i|^{t-1} = \frac{|O_i|^T - 1}{|O_i| - 1}$. *Q.E.D*

A T**-period joint policy in the sequence form** is an n-tuple $(p_1, p_2, \ldots, p_n)$ of policies in the sequence-form, one policy in the tuple per agent in I. The set of joint policies in the sequence-form $\times_{i \in I} X_i$ shall be denoted by X. The **weight** of a joint history j in a joint policy $(p_1, p_2, \ldots, p_n)$ in the sequence form is the product of the weights of the histories j_1 to j_n in respectively, the policies p_1 to p_n. That is, the weight of j in $(p_1, p_2, \ldots, p_n)$ is $\prod_{i \in I} p_i(j_i)$.

A T**-period** i**-reduced joint policy in the sequence form** is an (n- 1)-tuple of policies in the sequence form, one policy in the tuple per agent in I except for agent i. The set of i-reduced joint policies shall be denoted by X_{-i}. Given an i-reduced joint policy q_{-i} and a policy p_i of agent i, (p_i, q_{-i}) shall denote the joint policy in which the policies of all agents except i are according to q_{-i} and the policy of agent i is p_i.

3.4 Policy Constraints

A policy of agent i in the sequence-form can be found by solving the following linear program (LP):

$$\sum_{a \in A_i} x(a) = 1 \tag{3.4}$$

$$-x(h) + \sum_{a \in A_i} x(hoa) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{3.5}$$

$$x(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{3.6}$$

The LP merely implements the definition of a policy in the sequence-form. The LP contains one variable $x(h)$ for each history $h \in \mathcal{H}_i$ to represent the weight of h in the policy. A solution x^* to this LP constitutes a policy in the sequence-form. The policy is formed by the values of the x variables in the solution of the LP. So, the weight of a history h is $x^*(h)$. It may be a pure policy or it may be stochastic. The set of constraints (3.4)-(3.6) shall be called the **policy constraints** of agent i. They shall appear in all of the mathematical programs we present in the thesis.

To formulate this linear program in memory, we require space that is only exponential in the horizon. For each agent $i \in I$, the size of $\mathcal{H}_i$ is $\sum_{t=1}^{T} |A_i|^t |O_i|^{t-1}$. It is exponential in T. So the number of variables in the LP is exponential in T. The number of constraints in the LP (3.4)-(3.6) is $\sum_{t=0}^{T-1} |A_i|^t |O_i|^t$. So the number of constraints of the LP is also exponential in T.

However, despite the exponential size of the LP, it can be efficiently solved on account of the *sparsity* of the matrix of coefficients of the constraints of the LP. Let the number of constraints in the LP be denoted by m_i and let the number of variables in it be denoted by n_i. Let C_i denote the $m_i \times n_i$ matrix whose entries are the coefficients of the left-hand sides of the policy constraints (3.4)-(3.5). Thus, the entries of C_i are from the set $\{-1, 0, 1\}$. C_i is a sparse matrix since most of its entries are 0s. The following example illustrates the sparsity of the matrix C_i. Let $A_i = \{b, c\}$, $O_i = \{u, v\}$ and let $T = 2$. So, for this example, $n_i = 10$ and $m_i = 5$. The

system of policy constraints for this example is,

$$\begin{aligned} x_i(b) + x_i(c) &= 1 \\ -x_i(b) + x_i(bub) + x_i(buc) &= 0 \\ -x_i(c) + x_i(cub) + x_i(cuc) &= 0 \\ -x_i(b) + x_i(bvb) + x_i(bvc) &= 0 \\ -x_i(c) + x_i(cvb) + x_i(cvc) &= 0 \end{aligned}$$

Thereby, the matrix C_i is as follows,

$$C_i = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Notice that in the policy constraints of an agent, each variable is only constrained to be nonnegative whereas by the definition of a policy in sequence-form, the weight of a history must be in the interval [0, 1]. Does it mean that a variable in a solution to the policy constraints can assume a value higher than 1? Actually, the policy constraints are such that they prevent any variable from assuming a value higher than 1 as the following lemma shows.

Lemma 3.3. *In every solution x^* to* (3.4)-(3.6)*, for each $h \in \mathcal{H}_i$, $x^*(h) \in [0, 1]$.*

Proof: This can be shown by forward induction. Due to (3.4) and (3.6), for each $a \in A_i$, $x_i^*(a) \in [0, 1]$. Due to (3.5), there holds $\forall a \in A_i$ and $\forall o \in O_i$,

$$x^*(a) = \sum_{a' \in A} x^*(aoa') \tag{3.7}$$

Due to (3.4), $x^*(a) \in [0, 1]$. Hence, $\sum_{a' \in A_i} x^*(aoa') \in [0, 1]$. Hence, for each $a' \in A_i$, $x^*(aoa') \in [0, 1]$. Hence, we have the two facts: for each $a \in A_i$, $x^*(a) \in [0, 1]$ and for each $a \in A_i$, for each $o \in O_i$ and for each $a' \in A_i$, $x^*(aoa') \in [0, 1]$. Since an action is a history of length 1, the above two facts can be read as (setting $t = 1$), for each $h \in \mathcal{H}_i^t$, $x_i^*(h) \in [0, 1]$ and for each $h \in \mathcal{H}_i^t$, for each $o \in O_i$ and for each $a' \in A_i$, $x^*(hoa') \in [0, 1]$, in other words, for each $h' \in \mathcal{H}_i^{t+1}$, $x^*(h') \in [0, 1]$. Thereby, by induction this holds for $t = 2, 3, \ldots$.*Q.E.D*

In this thesis we shall often be interested in finding a pure policy in the sequence-form. (3.4)-(3.6) may find a pure policy, but it is not guaranteed to do so. If we wish to expressly find a pure policy, then we must make every variable in the LP a 0-1 variable. That is, we must replace (3.6) by

$$x(h) \in \{0, 1\}, \quad \forall h \in \mathcal{H}_i \tag{3.8}$$

However, not all variables in (3.4)-(3.6) need be turned into 0-1 variables in order to find a pure policy. As the following lemma shows, if we place 0-1 variables only variables representing terminal histories, due to the constraints of the LP, other variables automatically assume a value of either 0 or 1 in every solution of the LP.

Lemma 3.4. *If in* (3.4)-(3.6), (3.6) *is replaced by,*

$$x(h) \geq 0, \quad \forall h \in \mathcal{N}_i \tag{3.9}$$

$$x(h) \in \{0,1\}, \quad \forall h \in \mathcal{E}_i \tag{3.10}$$

then in every solution x^* to the resulting 0-1 MILP, for each $h \in \mathcal{H}_i$, $x^*(h) = 0$ or 1

Proof: We can prove this by backward induction. Let h be a history of of length T - 1. Due to (3.5), for each $o \in O_i$, there holds,

$$x^*(h) = \sum_{a \in A_i} x^*(hoa) \tag{3.11}$$

Since h is a history of length T - 1, each history hoa is is a terminal history. Due to Lemma 3.3, $x^*(h) \in [0,1]$. Therefore, the sum on the right hand side of the above equation is also in [0, 1]. But due to (3.10), each $x^*(hoa) \in \{0,1\}$. Hence the sum on the right hand side is either 0 or 1, and not any value in between. Ergo, $x^*(h) \in \{0,1\}$ and not any value in between. By this same reasoning, we can show that $x^*(h) \in \{0,1\}$, for every nonterminal history h of length T - 2, T - 3, ..., 1.*Q.E.D*

3.4.1 Example

An example of policy constraints with $A_i = \{b, c\}$, $O_i = \{u, v\}$ and $T = 3$, is as follows.

$$\begin{aligned}
x(b) + x(c) &= 1 \\
-x(b) + x(bub) + x(buc) &= 0 \\
-x(b) + x(bvb) + x(bvc) &= 0 \\
-x(c) + x(cub) + x(cuc) &= 0 \\
-x(c) + x(cvb) + x(cvc) &= 0 \\
-x(bub) + x(bubub) + x(bubuc) &= 0 \\
-x(bub) + x(bubvb) + x(bubvc) &= 0 \\
-x(buc) + x(bucub) + x(bucuc) &= 0 \\
-x(buc) + x(bucvb) + x(bucvc) &= 0 \\
-x(bvb) + x(bvbub) + x(bvbuc) &= 0 \\
-x(bvb) + x(bvbvb) + x(bvbvc) &= 0 \\
-x(bvc) + x(bvcub) + x(bvcuc) &= 0 \\
-x(bvc) + x(bvcvb) + x(bvcvc) &= 0 \\
-x(cub) + x(cubub) + x(cubuc) &= 0 \\
-x(cub) + x(cubvb) + x(cubvc) &= 0 \\
-x(cuc) + x(cucub) + x(cucuc) &= 0 \\
-x(cuc) + x(cucvb) + x(cucvc) &= 0 \\
-x(cvb) + x(cvbub) + x(cvbuc) &= 0 \\
-x(cvb) + x(cvbvb) + x(cvbvc) &= 0 \\
-x(cvc) + x(cvcub) + x(cvcuc) &= 0 \\
-x(cvc) + x(cvcvb) + x(cvcvc) &= 0
\end{aligned}$$

Every solution to this set of equations in which the variables are restricted to be nonnegative is a 3-period, possibly stochastic, policy in the sequence-form.

3.5 Value Of A Joint Policy

When a joint policy is executed, at the end of T steps, one terminal joint history can be said to have occurred. The probability with which a terminal joint history j occurs when a joint policy p is executed is the product of the probability with which the agents take the joint actions of the joint history (in other words, the weight of the joint history in p) and the probability that if the agents take the joint actions of the joint history, they receive the joint observations of the joint history. Therefore, the value of a joint policy in the sequence form can be defined in terms of the *values* of terminal joint histories and their weights in the joint policy where the value of a joint history is defined as the sum of the expected rewards obtained by the joint actions of the joint history times the probability with which the agents receive the joint observations of he joint history.

To be precise, the **value** of a joint policy $p \in X$ in the sequence-form for the given initial state α, denoted by $\mathcal{V}(\alpha, p)$, is defined as follows,

$$\mathcal{V}(\alpha, p) \quad = \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} p_i(j_i) \tag{3.12}$$

Here $\mathcal{R}(\alpha, j)$ denotes the value of joint history j. The value of a joint history is a property of the DEC-POMDP model. So it is independent of the joint policy. On the other hand, the weight of a joint history is independent of the DEC-POMDP model. It is solely a property of the joint policy. We now define the value of a joint history.

3.5.1 Value Of A Joint History

The **value** $\mathcal{R}(\beta, j)$ of a joint history j is defined as the product of two quantities:

$$\mathcal{R}(\alpha, j) \quad = \quad \Psi(\beta, j)\mathcal{S}(\alpha, j) \tag{3.13}$$

$\Psi(\alpha, j)$ is probability with which the agents receive the joint observations of j and $\mathcal{S}(\alpha, j)$ is the sum of expected reward obtained by the joint actions in j. $\Psi(\alpha, j)$ shall be called the **joint observations sequence probability** of j. The two quantities are defined and computed as follows.

Let $j = a^1, o^1, a^2, o^2, \ldots, o^{t-1}, a^t$ be a joint history of length t. Thereby, $\Psi(\alpha, j)$ is defined as the probability that the sequence of joint observations received by the agents till period t is $o^1, o^2, \ldots, o^{t-1}$ *if* the sequence of joint actions taken by them till period t - 1 is $a^1, a^2, \ldots, a^{t-1}$ *and* the initial state of the DEC-POMDP is α. That is,

$$\Psi(\beta, j) \quad = \quad \text{Prob.}(o^1, o^2, \ldots, o^{t-1}) | \alpha, a^1, a^2, a^{t-1})) \tag{3.14}$$

Note that the last joint action a^t of the joint history j is not involved in the definition of $\Psi(\alpha, j)$ this probability because this joint action is taken in period t *after* joint observation o^t has been received. This implies that the probability of a joint history of length 1 (that is, of an action) is 1. That is, $\Psi(\beta, a) = 1$ for each joint action a.

$\Psi(\beta, j)$ is computed as follows. If the joint history j occurs, then it means that the agents take a^1 in period 1 when the state of the process is α. Thereupon, they receive the joint observation o^1 with probability $\mathcal{T}(o^1|\alpha, a^1)$ at the beginning of the period 2. The agents take joint action a^2 in period 2 when the state of the process is α^1, the updated state of α given a^1 and o^1. Thus, in general, in period $k < t$, the agents take the joint action a^k when the state of the process is α^{k-1} and receive observation o^k with probability $\mathcal{T}(o^k|\alpha^{k-1}, a^k)$.

$\Psi(\alpha, j)$ is simply the product of the all the $\mathcal{T}$s. That is,

$$\Psi(\alpha, j) \quad = \quad \prod_{k=1}^{t-1} \mathcal{T}(o^k|\alpha^{k-1}, a^k) \tag{3.15}$$

where $\alpha^0 = \alpha$, and for each $k = 1$ to t - 1,

$$\mathcal{T}(o^k|\alpha^{k-1}, a^k) \quad = \quad \sum_{s\in S} \alpha^{k-1}(s) \sum_{s'\in S} \mathbb{P}(s, a^k, s')\mathbb{G}(a, s', o^k) \tag{3.16}$$

and for each $k = 1$ to t - 1,

$$\alpha^k(s') \quad = \quad \frac{\sum_{s\in S} \alpha^{k-1}(s)\mathbb{P}(s, a^k, s')\mathbb{G}(a, s', o^k)}{\mathcal{T}(o^k|\alpha^{k-1}, a^k)}, \quad \forall s' \in S \tag{3.17}$$

If for any k, $\mathcal{T}(o^k|\alpha^{k-1}, a^k) = 0$, then evidently $\Psi(\alpha, j) = 0$.

In computing $\Psi(\alpha, j)$, we compute the t - 1 states namely, α^1, α^2, ..., α^{t-1} and the t -1 probabilities, from $\mathcal{T}(o^1|\alpha^0, a^1)$ to $\mathcal{T}(o^{t-1}|\alpha^{t-2}, a^{t-1})$. Thereby, $\mathcal{S}(\alpha, j)$ is simply,

$$\mathcal{S}(\alpha, j) \quad = \quad \sum_{k=1}^{t} \sum_{s\in S} \alpha^{k-1}(s)R(s, a^k) \tag{3.18}$$

Just as the value of a joint policy in the canonical form at a state interior to the simplex $\Delta(S)$ can be expressed as a convex combination of the values of the joint policy at the corners of $\Delta(S)$ (i.e., the members of S), the value of a joint history at an interior state can be similarly expressed as a convex combination of the values of the joint history at states in S. That is, for any joint history j and for any $\beta \in \Delta(S)$, there holds,

$$\mathcal{R}(\beta, j) \quad = \quad \sum_{s\in S} \beta(s)\mathcal{R}(s, j) \tag{3.19}$$

That (3.19) holds can be proved in several ways follows. A simple proof is as follows.

Consider the 2-agent joint history (h_1, h_2) = (buc, cvb) of length 3. The joint actions are (b, c) and (c, b) and the joint observation is (u, v). Let (σ_1, σ_2) be a 2-period joint policy in the canonical form for which $\sigma_1(\varnothing) = b$, $\sigma_1(u) = c$, $\sigma_2(\varnothing) = c$ and $\sigma_2(v) = b$. Now, assume that the state transition function $\mathbb{P}$ and the joint observation function $\mathbb{G}$ are such that the probability of receiving any of the other three joint observations - (u, u), (v, u) and (v, v) - when the agents take the joint action (b, c) in α is 0 (the set of observations of the two agents is assumed to be $\{u, v\}$). Therefore, σ_1 can assign any action to the observation v and σ_2 can assign any action to the observation u without affecting the value of the joint policy (σ_1, σ_2). Therefore, the value

of (σ_1, σ_2) equals the value of the joint history (h_1, h_2). In effect, (h_1, h_2) is also a 2-period joint policy (in the canonical form). Since the value of (σ_1, σ_2) at α can be expressed as a convex combination of its values at the states in S, we can similarly express the value of (h_1, h_2) at α can also be expressed as a convex combination of its values at the states in S.

3.6 Nonlinear Program

From the definitions of a policy in the sequence form and the value of a joint policy in the sequence form, it follows that we can find an optimal joint policy by solving the following nonlinear program (NLP):

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{3.20}$$

Subject To,

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{3.21}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{3.22}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{3.23}$$

We shall henceforth refer to this NLP as **NLP1**. This NLP has, for each agent $i \in I$ and for each history $h \in \mathcal{H}_i$ of agent i a variable $x_i(h)$. The constraints of the program ensure that each x_i vector is a (possibly stochastic) policy in the sequence-form. The vectors together constitute a joint policy. The objective function of the NLP is to maximize the value of the joint policy found. Therefore, a global maximum point[7] of the NLP $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an optimal joint policy.

The constraints of the program form a convex set, but the objective function is not concave. Finding a global maximum point of a nonconcave function is a very difficult problem. There are no generalized methods that guarantee finding it. Most methods guarantee finding a local maximum point, but even that is an NP-hard problem. A local maximum point of this NLP is a locally optimal joint policy, which as defined in Chapter 1, can have a value that is much lower than that of the optimal joint policy. An evident, but inefficient method to find a global maximum point is to evaluate *all* the extreme points of set of feasible solutions of the program since it is known that every global as well as local maximum point of a nonconcave function lies is an extreme point of the set of feasible solutions of the program. This is in an inefficient method because there is no test that tells is if an extreme point is a local maximum point or a global maximum point. Hence, unless all extreme points are evaluated, we cannot be sure of having

[7] A function g defined on a convex set M is said to be *concave* if for every w, $w' \in M$, and every real number α, $0 \leq \alpha \leq 1$, there holds,

$$g(\alpha w + (1-\alpha)w') \geq \alpha g(w) + (1-\alpha) g(w') \tag{3.24}$$

Geometrically, a function is concave if the line joining any two points on its graph lies nowhere above its graph. Given a function g defined over a convex set M, $w^* \in M$ is a *local maximum point* of g over M if there exists an $\epsilon > 0$ such that for all $w \in M$ within a distance of ϵ from w^* (i.e., $|w - w^*| < \epsilon$), $g(w^*) \geq g(w)$. Further, $w^* \in M$ is a *global maximum point* of g over M if for all $w \in M$, $g(w^*) \geq g(w)$.

obtained a global maximum point. The set of feasible solutions to the NLP is X. The set of extreme points of this set is $\hat{X}$, the set of pure joint policies, whose number is doubly exponential in T and exponential in n. So enumerating the extreme points for this NLP is intractable.

3.7 Summary

In this chapter we have described the sequence-form of a finite horizon policy. We have defined a linear program that finds a policy in the sequence form. We have defined the value of a joint policy in the sequence form. Finally, we have defined a **NLP1**, a nonlinear program whose optimal solution yields an optimal joint policy. But existing methods are not capable of always finding an optimal solution to **NLP1**. In the next chapter, we shall linearize **NLP1** through two different methods to two different equivalent 0-1 mixed integer linear programs (MILPs). An optimal solution of either of these MILPs is an optimal joint policy. Methods that always find an optimal solution to a 0-1 MILP exist. Note that **NLP1** requires space to be formulated that is exponential in T and linear in n. The 0-1 MILPs on the other hand require space that is exponential in T and exponential in n. Thus, in conceiving a viable alternative to **NLP1**, we incur a cost in terms of increased space requirement.

Before moving onto the next chapter, it shall be worth our while to consider why the approach we embrace for finding an optimal joint policy - represented in a rudimentary form by **NLP1** - is potentially advantageous over existing algorithms. Existing algorithms find an optimal joint policy in the canonical form. As we have seen, every policy of an agent in the canonical defines a distribution of weights over the agent's set of histories. Thus, an optimal joint policy σ^* in the canonical form produces two quantities: a value $\mathcal{V}(\alpha, \sigma^*)$ (i.e., the expected reward obtained if that policy is executed) and $p(\sigma^*)$, a tuple of distributions over the agents' sets of histories. However, our sole purpose in finding σ^* - as far as control of the DEC-POMDP is concerned - is to produce $\mathcal{V}(\sigma^*)$; the production of $p(\sigma^*)$ can be considered to be purely incidental. In fact, existing algorithms are not even aware that in finding σ^*, they are also finding $p(\sigma^*)$. We may say that existing algorithms *explicitly* find optimal joint policy in the canonical form and *implicitly* find the tuple of weight distributions over sets of histories produced by the joint policy.

But we can turn this argument on its head. We can exchange the quantity explicitly found for the one implicitly found. Since our only purpose is to maximize value (irrespective of the manner in which it is produced), instead of finding an optimal joint policy in the canonical form, we can find an "optimal" tuple p^* of weight distributions over the sets of agents' histories, i.e., a tuple of weight distributions which maximizes value. If we are able to find p^*, then we have effectively achieved optimal control of the DEC-POMDP since the weight distribution of each agent in the tuple (assuming that is obeys the policy constraints of the agent) is a *de facto* control policy. Such a policy is in fact called a policy in the sequence form. Moreover, if one wishes, a policy in the sequence form can be transformed into an equivalent (possibly mixed) policy in the canonical form. Thus, our approach consists of *explicitly* finding an optimal tuple of weight distributions over sets of histories (i.e., an optimal joint policy in the sequence form) and *implicitly* finding the joint policy in the canonical form that is responsible for producing the tuple of weight distributions.

As to the advantage of our approach, an optimal joint policy in the sequence form is a tuple of weight distributions over the agents' set of histories. Finding an optimal joint policy in the

sequence form may be easier than finding an optimal joint policy if we take into account the fact that the number of histories of each agent is "only" exponential in the horizon while the number of policies of an agent in the canonical form is doubly exponential in the horizon. Therefore, an optimal joint policy in the sequence form can be thereby found by formulating an appropriate mathematical program such as **NLP1** that requires relatively little space (and hopefully also relatively little time), compared to the space and time requirements of existing algorithms.

Chapter 4

A Combinatorial Optimization Approach

4.1 Introduction

We have seen in Chapter 3 that we can find an optimal joint policy for a finite horizon n-agent DEC-POMDP by solving the nonlinear program (NLP) **NLP1** given in Section (3.6). A globally maximum solution $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ of this program constitutes an optimal T-period joint policy (in the sequence-form). A locally maximum solution to the program constitutes a locally optimal joint policy. As explained in Chapter 3, the objective function of **NLP1** is not concave and thereby upon solving this NLP, we are only guaranteed to find a locally optimal joint policy. In this chapter, we shall linearize (3.20)-(3.23) to two slightly different 0-1 integer linear programs (ILPs) using two slightly different techniques. Linearization involves converting the nonlinear function to a linear function, for which we are required to introduce additional variables and constraints to the program. A solution of either of these two ILPs is a pure optimal n-agent joint policy.

The two 0-1 ILPs are both conceived using a simple property consisting of the following two facets:

(1) In every DEC-POMDP, there exists an optimal T-period joint policy that is pure.

(2) The support of a pure T-period joint policy is of a fixed size. Recall that the support of a pure joint policy is the set of joint histories that receive a weight of 1 in it. A pure joint policy is thereby just a subset of the set of joint histories.

These facts move the DEC-POMDP problem directly into the ambit of *combinatorial optimization* [Kle80]. Problems of combinatorial optimization typically involve finding a subset of a given finite set that satisfies some criteria. Such problems are typically treated using 0-1 integer linear programming. In fact the very field of 0-1 integer linear programming could be said to have sprung to solve such problems without having to enumerate all possible subsets. When applied to a problem of combinatorial optimization, the 0-1 variables in a 0-1 ILP are meant to identify the members of the required subset; members of the set whose variables have value 1 in the solution are accepted in the subset; those whose variables have value 0 in the solution are rejected.

Several important problems in computer science such as the knapsack problem, the traveling salesman problem, the quadratic assignment problem (QAP) etc [PS82] belong to the domain combinatorial optimization. The (discrete) Team Decision Problem [TA85], described in Chapter 1, can also be considered as a problem of combinatorial optimization. The knapsack problem is a particularly simple example. Here, we are given n integers, d_1, d_2, ..., d_n and we required to find a combination of these integers whose sum equals a given integer d. This problem can be solved by enumerating all combinations of all sizes of the n integers. But in practice, 0-1 integer linear programming is much more efficient than an enumeration of combinations. The knapsack problem is solved by the following 0-1 ILP

$$\sum_{i=1}^{n} d_i x_i = d \tag{4.1}$$

$$x_i \in \{0,1\}, \quad i = 1, 2, \ldots, n \tag{4.2}$$

The required combination (if one exists) consists of each integer d_i for which $x_i^* = 1$. Thus, in the knapsack problem, a subset of the n integers is found; the size of this subset is not fixed.

Due to facts (1) and (2), to find an optimal T-period joint policy, we are required to find a subset of the set of joint histories that satisfies certain criteria (it should maximize value and we should be able to infer individual agent policies from the subset). By treating the DEC-POMDP problem as a problem of combinatorial optimization, we can exploit techniques and heuristics developed in this domain. In particular, the DEC-POMDP problem is structurally quite similar to the QAP. The QAP in its basic formulation is a nonlinear program just as the DEC-POMDP problem is the NLP (3.20)-(3.23). The QAP must be converted to a 0-1 integer linear program if we are to find its globally optimal solution. This conversion is, of course, nontrivial. The principle behind the conversion of the QAP from a nonlinear program to a 0-1 ILP can also be applied to convert the DEC-POMDP problem from its basic nonlinear program to a 0-1 ILP. We present a 0-1 ILP for solving the DEC-POMDP problem that is inspired by this principle in Section (4.2). However, this 0-1 ILP can be improved upon in terms of space requirement. The principle which achieves the conversion can be substituted by an other principle, which results in a much smaller 0-1 ILP. We present this improved 0-1 ILP in Section (4.3). Thus, in the two sections (4.2) and (4.3), we present two 0-1 ILPs for solving the DEC-POMDP problem, the one presented in the latter being an improvement over the one presented in the former.

Finally, we shall also show that how each of the two 0-1 ILPs can be partially relaxed to an equivalent 0-1 mixed integer linear program (MILP). The relaxations are faster to solve than the ILPs since they have lesser 0-1 variables.

4.1.1 The Quadratic Assignment Problem

The QAP is similar to the 2-agent (discrete) Team Decision Problem (TDP) described in Chapter 1. In the QAP, we are given two sets I' and J' each of size m. We are given for each $i, i' \in I'$ and for each $j, j' \in J'$, a number $c_{iji'j'}$ called the cost of forming the pair (i, j) *and* the pair (i', j'). Our objective is to form m pairs from the m^2 possible pairs in $I' \times J'$ whose sum of costs is minimum. The conditions for forming the m pairs are that each member of I' must be in exactly one pair and each member of J' must be in exactly one pair. The QAP corresponds

to the following 0- 1 integer nonlinear program (INLP):

$$\text{Minimize} \quad \sum_{i,i' \in I'} \sum_{j,j' \in J'} c_{iji'j'} x(i,j) x(i',j') \tag{4.3}$$

Subject To:

$$\sum_{j \in J'} x(i,j) = 1, \quad \forall i \in I' \tag{4.4}$$

$$\sum_{i \in I'} x(i,j) = 1, \quad \forall j \in J' \tag{4.5}$$

$$x(i,j) \in \{0,1\}, \quad \forall i \in I', \forall j \in J' \tag{4.6}$$

The program contains one variable $x(i,j)$ for every pair $(i,\ j) \in I' \times J'$; if $x^*(i,j) = 1$, then the pair is formed, otherwise it is not. The constraints ensure that the conditions for forming the pairs are respected. The objective function of this program being nonconvex, we are not guaranteed to find a globally optimal solution of the program unless we modify (linearize) the program. In other words, the subset of $I' \times J'$ of pairs we obtain by directly solving this program may not be the one with the smallest total cost.

As shown in [Law63], the 0-1 INLP (4.3)-(4.6) can be linearized to the following 0-1 ILP by using one variable $z(i,j,i',j')$ for every pair of pairs $(i,\ j)$ and $(i',\ j')$, and by adding an extra set of constraints:

$$\text{Minimize} \quad \sum_{i,i' \in I'} \sum_{j,j' \in J'} c_{iji'j'} z(i,j,i',j') \tag{4.7}$$

Subject To:

$$\sum_{j \in J'} x(i,j) = 1, \quad \forall i \in I' \tag{4.8}$$

$$\sum_{i \in I'} x(i,j) = 1, \quad \forall j \in J' \tag{4.9}$$

$$\sum_{i,i' \in I'} \sum_{j,j' \in J'} z(i,j,i',j') = m^2 \tag{4.10}$$

$$x(i,j) + x(i',j') - 2z(i,j,i',j') \geq 0, \quad \forall i,i' \in I, \forall j,j' \in J' \tag{4.11}$$

$$x(i,j) \in \{0,1\}, \quad \forall i \in I', \forall j \in J' \tag{4.12}$$

$$z(i,j,i',j') \in \{0,1\}, \quad \forall i,i' \in I', \forall j,j' \in J' \tag{4.13}$$

For each pair of pairs $(i,\ j)$, $(i',\ j')$ if $z^*(i,j,i',j') = 1$, then the pairs $(i,\ j)$ and $(i',\ j')$ are both formed. A solution to this program, obtainable through the BB method, gives us the required subset of $I' \times J'$ with the least sum of costs. As proved in [Law63], the constraints (4.10) and (4.11) guarantee that the following holds true for every pair of pairs $(i,\ j)$ and $(i',\ j')$ in every solution $(x^*,\ z^*)$ to this program:

$$z^*(i,j,i',j') = 1 \quad \Leftrightarrow \quad x^*(i,j) = 1 \quad \text{and} \quad x^*(i',j') = 1 \tag{4.14}$$

Thus, given a solution $(x^*,\ z^*)$ to (4.7)-(4.13), x^* is a solution to (4.3)-(4.6).

We can see that the 0-1 INLP (4.3)-(4.6) and the 0-1 INLP (1.2)-(1.4) for the TDP given in Chapter 1, represent more or less the same problem. Therefore, (1.2)-(1.4) can be converted to a 0-1 ILP in the same manner as (4.3)-(4.6) has been converted to a 0-1 ILP.

4.2 A 0-1 Integer Linear Program

We can see that **NLP1** is structurally quite similar to (4.3)-(4.6). In this section, we shall use a linearization technique similar to the one employed to convert the QAP to a 0-1 ILP to convert the NLP to a 0-1 ILP. A solution of the 0-1 ILP shall give an optimal joint policy.

Since the objective function of the program is the only nonlinear quantity in the entire program, we shall use a 0-1 variable $z(j)$ for every terminal joint history $j \in H^T$. This means that we replace each term $\prod_{i \in I} x_i(j_i)$ in the objective function by the single variable $z(j)$. This shall allow us to linearize the objective function of **NLP1**. That is, the nonlinear objective function,

$$\text{maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{4.15}$$

can now be rewritten as,

$$\text{maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.16}$$

Note that $j = (j_1, j_2, \ldots, j_n)$. Now, we have managed to linearize the objective function, but it does not have any variables in common with the constraints. In effect we are now using double variables for the same quantity (the variable $z(j)$ and the product of the variables $\prod_{i \in I} x_i(j_i)$ now represents the same quantity). It is necessary that the values of the two be the same in every solution. In other words, in every solution to the resulting program, for each terminal joint history j, the following double implication must hold:

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.17}$$

In order to ensure this double implication, we require that every variable in the program be a 0-1 variable and we add the following constraints to the NLP:

$$\sum_{i \in I} x_i(j_i) - n z(j) \quad \geq \quad 0, \quad \forall j \in \mathcal{E} \tag{4.18}$$

$$\sum_{j \in \mathcal{E}} z(j) \quad = \quad \prod_{i \in I} |O_i|^{T-1} \tag{4.19}$$

Due to (4.18) (and the fact that the variables are all 0-1 variables), the implication,

$$z^*(j) = 1 \quad \Rightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.20}$$

necessarily holds. The effect of (4.19) (in conjunction with the other constraints) is that the implication holds in the other direction as well. Note that according to Lemma (3.1) of Chapter 3, the number of terminal histories that receive a weight of 1 in a pure policy of agent i is $|O_i|^{T-1}$. Thereby, the number of terminal joint histories that receive a weight of 1 in a pure joint policy is $\prod_{i \in I} |O_i|^{T-1}|$. Hence we add the constraints (4.19).

With these additions, **NLP1** is transformed to the following 0-1 ILP:

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.21}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{4.22}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{4.23}$$

$$\sum_{i \in I} x_i(j_i) - nz(j) \geq 0, \quad \forall j \in \mathcal{E} \tag{4.24}$$

$$\sum_{j \in H^T} z(j) = \prod_{i \in I} |O_i|^{T-1}| \tag{4.25}$$

$$x_i(h) \in \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{4.26}$$

$$z(j) \in \{0,1\}, \quad \forall j \in \mathcal{E} \tag{4.27}$$

We shall henceforth refer to the 0-1 ILP (4.21)-(4.27) as **ILP1**.

4.2.1 Validity Of The Program

The following lemma shows that the constraints (4.24) and (4.25) together ensure that values assigned to z variables are not inconsistent with those assigned to the x variables.

Lemma 4.1. *Every solution (x^*, z^*) to* **ILP1** *satisfies the following condition for every terminal joint history $j \in \mathcal{E}$,*

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.28}$$

Proof: Let (x^*, z^*) denote a solution to **ILP1**. Let,

$$S(z) = \{j \in \mathcal{E} | z^*(j) = 1\} \tag{4.29}$$

$$S(x_i) = \{h \in \mathcal{E}_i | x_i^*(h) = 1\}, \quad \forall i \in I \tag{4.30}$$

$$S(x) = \{j \in \mathcal{E} | x_i^*(j_i) = 1, \forall i \in I\} \tag{4.31}$$

If we show that $S(x)$ is identical to $S(z)$, then the statement of the lemma is proved.

Since each variable is a 0-1 variable, due to (4.25), the following implication clearly holds for each terminal joint history j,

$$z^*(j) = 1 \quad \Rightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.32}$$

For each agent i, each variable in x_i is a 0-1 variable. Since each x_i is required to satisfy the policy constraints of agent i, each x_i^* is a pure policy. Therefore, $|S(x)| = \prod_{i \in I} |O_i|^{T-1}$. Since each z variable is also a 0-1 variable, due to (4.24), there holds $|S(z)| = \prod_{i \in I} |O_i|^{T-1}$. So, the two sets $S(x)$ and $S(z)$ are of the same size.

Now, assume that $S(x) \neq S(z)$. If this assumption is true, then since the sizes of the two sets are equal, it means that there exists at least one pair of terminal joint histories $j, j' \in \mathcal{E}$ such that (i) $j \in S(z)$ but $j \notin S(x)$ and (ii) $j' \notin S(z)$ but $j' \in S(x)$. But (i) is not possible since if this were the case, it would mean that for j,

$$z^*(j) = 1 \quad \nRightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.33}$$

This is a clear contradiction, since we have proved above that this implication is true for every terminal joint history. So, if assumption $S(z) \neq S(x)$ is to be true, then the two sets can be different only in the manner of (ii), that is, there must exist a terminal joint history j' such that $j' \notin S(z)$ but $j' \in S(x)$. But then, it would mean $|S(z)| < |S(x)|$, which is not true since we have proved that the two sets are of the same size. Hence, the assumption that $S(x) \neq S(z)$ is false. This proves the statement of the lemma. *Q.E.D*

We therefore have the following result.

Theorem 4.1. *Given a solution (x^*, z^*) to* **ILP1**, *$x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an optimal joint policy*

Proof: Let (x^*, z^*) denote a solution to **ILP1**. Since for each agent $i \in I$, x_i^* is a policy, therefore $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is a pure joint policy. Due to Lemma (4.1), in every solution (x^*, z^*) of the program, there holds,

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z^*(j) \quad = \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i^*(j_i) \tag{4.34}$$

The quantity on the right-hand side is the value of x^*. Since we have maximized the quantity on the left-hand side, we have effectively maximized the quantity on the right-hand side. Hence x^* is a joint policy with maximum value. *Q.E.D*

Note that for the linearization technique used to convert the NLP to a 0-1 ILP, it is not sufficient that we convert the objective function to a linear function by using variables for terminal joint histories; it is also necessary that the variables for terminal joint histories and the variables for terminal histories be 0-1 variables. Only then does the implication (4.32) which is central to the proof of Lemma (4.1) hold. If we used continuous variables for either histories and/or terminal joint histories in **ILP1**, Lemma (4.1) would not be true, which would mean that it is possible that in some solution (x^*, z^*) to the program, for some terminal joint history j that,

$$z^*(j) \quad \neq \quad \prod_{i \in I} x_i^*(j_i) \tag{4.35}$$

implying that,

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z^*(j) \quad \neq \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i^*(j_i) \tag{4.36}$$

Thereby, a solution to the program would not be guaranteed to be an optimal joint policy.

On the other hand, we can use 0-1 variables for histories of the agents only because of the fact that in every DEC-POMDP there exists an optimal joint policy that is pure. Thus, in conceiving the linearization technique, we are exploiting indirectly the simple fact that in every DEC-POMDP there exists an optimal joint policy that is pure. In converting the NLP to **ILP1**, we have managed to construct an algorithm finding an optimal joint policy, but our space requirement has gone up considerably. The number of variables and constraints in the NLP is exponential only in T, while the number of variables and constraints in **ILP1** is exponential in T as well as in n. In the next section, we conceive a different the 0-1 ILP for the DEC-POMDP problem whose number of constraints is exponential only in T even though it has the same number of variables as **ILP1**.

4.3 An Improved 0-1 Integer Linear Program

The 0-1 ILP we present in this section is also exploits the fact that in every DEC-POMDP there is an optimal joint policy that is pure. However, it exploits this fact in a slightly different manner than **ILP1**. The linearization technique employed by the 0-1 ILP presented in this section is described as follows.

According to Lemma (3.1) of Chapter 3, the number of terminal histories of length t of agent i that are in the support of a pure policy of the agent is $|O_i|^{T-1}$. Thereby, the number of terminal joint histories of length t that are in the support of a pure joint policy is $\prod_{i\in I}|O_i|^{T-1}$. Hence, the number of terminal joint histories of which a history h of length t of agent i is a part of, and which are in the support of a pure joint policy is $\frac{\prod_{k\in I}|O_k|^{T-1}}{|O_i|^{T-1}} = \prod_{k\in I\setminus\{i\}}|O_k|^{T-1}$. Therefore, the disjunction "either a history of length t agent i is in the support of the policy of the agent in the joint policy *or* it is not" is equivalent to the disjunction "either the number of joint histories of which the history is a part of and which are in the support of the joint policy is $\prod_{k\in I\setminus\{i\}}|O_k|^{T-1}$ *or* it is 0".

The equivalence between the two disjunctions can be exploited in the following manner. In every solution x^* to **ILP1**, for every terminal history h of agent i, the following equivalence of disjunctions exists,

$$x_i^*(h) = 1 \quad \Leftrightarrow \quad \sum_{j\in\mathcal{E}:j_i=h}\prod_{i\in I}x_i^*(j_i) = \prod_{k\in I\setminus\{i\}}|O_k|^{T-1} \tag{4.37}$$

$$x_i^*(h) = 0 \quad \Leftrightarrow \quad \sum_{j\in\mathcal{E}:j_i=h}\prod_{i\in I}x_i^*(j_i) = 0 \tag{4.38}$$

Now suppose that the constraints (4.24) were removed from **ILP1**. Then, we would not be able to ensure that for every terminal joint history j, there holds,

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.39}$$

On the other hand, due to the above equivalence of disjunctions, if we replace (4.25) in **ILP1** by the following set of constraints,

$$\sum_{j'\in\mathcal{E}_{-i}} z(h,j') \quad = \quad \prod_{k\in I\setminus\{i\}}|O_k|^{T-1}x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.40}$$

then in every solution (x^*, z^*) of the resulting 0-1 ILP, for every terminal joint history j, the aforementioned double implication would hold (a following lemma proves this). Note that (h, j') denotes the joint history j in which $j_i = h$ and $j_{-i} = j'$.

Thus, by replacing (4.25) in **ILP1** by the set of constraints (4.40), we obtain the following 0-1 ILP:

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.41}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{4.42}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{4.43}$$

$$\sum_{j' \in H^T_{-i}} z(h, j') = \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.44}$$

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{4.45}$$

$$x_i(h) \in \{0, 1\}, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{4.46}$$

$$z(j) \in \{0, 1\}, \quad \forall j \in \mathcal{E} \tag{4.47}$$

We shall henceforth refer to the 0-1 ILP (4.41)-(4.47) as **ILP2**.

4.3.1 Validity Of The Program

The following lemma and theorem prove that this program finds an optimal joint policy.

Lemma 4.2. *Every solution (x^*, z^*) to* **ILP2** *satisfies the following condition for every terminal joint history $j \in \mathcal{E}$,*

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.48}$$

Proof: The proof is similar to the proof of Lemma (4.1). Let (x^*, z^*) denote a solution to **ILP2**. Let,

$$S(z) = \{j \in \mathcal{E} | z^*(j) = 1\} \tag{4.49}$$

$$S(x_i) = \{h \in \mathcal{E}_i | x_i^*(h) = 1\}, \quad \forall i \in I \tag{4.50}$$

$$S(x) = \{j \in \mathcal{E} | x_i^*(j_i) = 1, \forall i \in I\} \tag{4.51}$$

If we show that $S(x)$ is identical to $S(z)$, then the statement of the lemma is proved.

Since each variable in the program is a 0-1 variable, due to (4.44), the following implication clearly holds for each terminal joint history j,

$$z^*(j) = 1 \quad \Rightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.52}$$

For each agent i, each variable in x_i is a 0-1 variable. Since each x_i is required to satisfy the policy constraints of agent i, each x_i^* is a pure policy. Therefore, $|S(x)| = \prod_{i \in I} |O_i|^{T-1}$. Since each z variable is also a 0-1 variable, due to (4.44), there holds $|S(z)| = \prod_{i \in I} |O_i|^{T-1}$. So, the two sets $S(x)$ and $S(z)$ are of the same size. The remainder of the proof is identical to the proof of Lemma (4.1). *Q.E.D*

We therefore have the following result.

Theorem 4.2. *Given a solution (x^*, z^*) to* **ILP2**, *$x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an optimal joint policy*

Proof: The proof is analogous to the proof of Theorem (4.1). We use Lemma (4.2) in place of Lemma (4.1).*Q.E.D*

We can see that in terms of space, **ILP2** is an improvement over **ILP1**. While the number of variables in the two programs is the same, the number of constraints in **ILP2** is exponential in T while the number of constraints in **ILP1** is exponential in T and in n.

4.4 Equivalent Relaxations

In this section, we describe how **ILP1** and **ILP2** can be each relaxed to equivalent 0-1 mixed integer linear programs. Relaxing an ILP means allowing some variables in it to be continuous variables. We say that a MILP is **equivalent** to an ILP if every solution to the former is also a solution to the latter. Each solution to **ILP1** or to **ILP2** is a pure optimal joint policy because every variable in the each program is a 0-1 variable. In the following we show that by even relaxing some variables, the same holds, that is, their solutions are still pure optimal joint policies.

First, note that each program can be relaxed to a 0-1 MILP by removing the 0-1 integer constraints from variables representing non-terminal histories. That is, in each program, we replace

$$x_i(h) \in \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{4.53}$$

by,

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{4.54}$$

$$x_i(h) \in \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.55}$$

As proved in Lemma (3.4) in Chapter 3, for any n_i-vector x_i that satisfies (4.54), (4.55) and the policy constraints,

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{4.56}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{4.57}$$

the value of $x_i(h)$, for every $h \in \mathcal{H}_i$, is either 0 or 1.

ILP1 cannot be further relaxed. We cannot the relax z variables in it. So its relaxation yields the following 0-1 MILP:

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.58}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{4.59}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{4.60}$$

$$\sum_{i=1}^{n} x_i(j_i) - nz(j) \geq 0, \quad \forall j \in \mathcal{E} \tag{4.61}$$

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{4.62}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{4.63}$$

$$x_i(h) \in \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.64}$$

$$z(j) \in \{0,1\}, \quad \forall j \in \mathcal{E} \tag{4.65}$$

We shall henceforth refer to the 0-1 MILP (4.58)-(4.65) as **MILP1**.

In **ILP2**, we can go even further; we can relax even the z variables. Apart from relaxing the$x$ variables as in **MILP1**, in this program, we can replace

$$z(j) \in \{0,1\}, \quad \forall j \in \mathcal{E} \tag{4.66}$$

by,

$$z(j) \in [0,1], \quad \forall j \in \mathcal{E} \tag{4.67}$$

As the lemma that follows shows, the value of each variable z in any solution of the changed program is still 0 or 1. The 0-1 MILP relaxation of **ILP2** is as follows:

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{4.68}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{4.69}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{4.70}$$

$$\sum_{j' \in H^T_{-i}} z(h, j') = \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.71}$$

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{4.72}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{4.73}$$

$$x_i(h) \in \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{4.74}$$

$$z(j) \in [0,1], \quad \forall j \in \mathcal{E} \tag{4.75}$$

We shall henceforth refer to the 0-1 MILP (4.68)-(4.75) as **MILP2**.

4.4.1 Equivalence Of The Relaxations

MILP1 is identical to **ILP1** in every respect except for the fact that variables for nonterminal histories in the former are not 0-1 variables, and in the latter they are. However, due to Lemma (3.4), in every solution to **MILP1**, the value of every variable representing a nonterminal history is either 0 or 1; in effect it is a 0-1 variable. Hence, **MILP1** is equivalent to **ILP1**. This means that every solution to **MILP1** is also an optimal joint policy.

Similarly, **MILP2** is identical to **ILP2** in all respects but two. Variables for nonterminal histories are not 0-1 variables in the former while in the latter they are; variables for terminal joint histories (the z variables) are not 0-1 variables in the former while in latter they are. Due to Lemma (3.4), we know that the value of every variable representing a nonterminal history in **MILP2** is either 0 or 1. So, to establish the equivalence of **MILP2** to **ILP2**, we need to show that in every solution to the former, the value of every z variable is either 0 or 1.

Lemma 4.3. *In every solution (x^*, z^*) to* **MILP2***, for each $j \in \mathcal{E}$, $z^*(j)$ is either 0 or 1.*

Proof: Let (x^*, z^*) be a solution **MILP2**. Let,

$$S(z) = \{j \in \mathcal{E} | z^*(j) > 0\} \tag{4.76}$$

$$S(x_i) = \{h \in \mathcal{E}_i | x_i^*(h) = 1\}, \quad \forall i \in I \tag{4.77}$$

$$S_i(z, j') = \{j \in \mathcal{E} | j_{-i} = j', z^*(j) > 0\}, \quad \forall i \in I, \forall j' \in \mathcal{E}_{-i} \tag{4.78}$$

Now, due to (4.72) and (4.75), $|S(z)| \geq \prod_{i \in I} |O_i|^{T-1}$. By showing, that $|S(z)| \leq \prod_{i \in I} |O_i|^{T-1}$, we shall establish that $|S(z)| = \prod_{i \in I} |O_i|^{T-1}$. Then due to the upper bound of 1 on each z variable, the implication will be that $z^*(j)$ is 0 or 1 for each terminal joint history j thus proving the statement of the lemma.

Note that by Lemma (3.4) of Chapter 3, for each agent i, x_i^* is a pure policy. Therefore, we have that $|S_i(x)| = |O_i|^{T-1}$. This means that in the set of constraints (4.71), an i-reduced terminal joint history $j' \in \mathcal{E}_{-i}$ will appear on the right hand side not more than $|O_i|^{T-1}$ times when in the left hand side, we have $x_i^*(h) = 1$. Thus, $\forall j' \in \mathcal{E}_{-i}$,

$$|S_i(z, j')| \leq |O_i|^{T-1} \tag{4.79}$$

Now, we know that for each agent i and for each history $h \in \mathcal{H}_i$, $x_i^*(h)$ is either 0 or 1 since x_i^* is a pure policy. So, given an i-reduced terminal joint history j', $\prod_{k \in I \setminus \{i\}} x_k^*(j'_k)$ is either 0 or 1. Secondly, due to (4.71), the following implication clearly holds for each terminal joint history j,

$$z^*(j) > 0 \Rightarrow x_i^*(j_i) = 1, \quad \forall i \in I \tag{4.80}$$

Therefore, we obtain,

$$|S_i(z, j')| \leq |O_i|^{T-1} \tag{4.81}$$

$$= |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) \tag{4.82}$$

Therefore,

$$\begin{aligned}
\sum_{j' \in \mathcal{E}_{-i}} |S_i(z, j')| &\leq \sum_{j' \in \mathcal{E}_{-i}} |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) && (4.83)\\
&= |O_i|^{T-1} \sum_{j' \in \mathcal{E}_{-i}} \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) && (4.84)\\
&= |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} \sum_{h' \in \mathcal{E}_k} x_k^*(h') && (4.85)\\
&= |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} && (4.86)\\
&= \prod_{j \in I} |O_j|^{T-1} && (4.87)
\end{aligned}$$

Since, $\bigcup_{j' \in \mathcal{E}_{-i}} S_i(z, j') = S(z)$, there holds, $\sum_{j' \in \mathcal{E}_{-i}} |S_i(z, j')| = |S(z)|$. Hence,

$$|S(z)| \leq \prod_{j \in I} |O_j|^{T-1} \quad (4.88)$$

Thus the statement of the lemma is proved. *Q.E.D*

Therefore, **MILP2** is equivalent to **ILP2** and thereby every solution to it is an optimal joint policy.

4.4.2 The Branch And Bound Method

In order to understand the virtue of relaxing a 0-1 ILP to an equivalent 0-1 MILP, it is important to understand the manner in which a 0-1 ILP or 0-1 MILP is solved. The principle method for solving 0-1 (M)ILPs is the *branch and bound* (BB) method [Fle87], first proposed in 1960 by Land and Doig [LD60].

The number of possible solutions of a 0-1 ILP or 0-1 MILP is essentially a function of the number of 0-1 variables in the program. If there are n 0-1 variables in a 0-1 (M)ILP, there are 2^n ways in which the 0-1 variables can be assigned values of 0s and 1s. Therefore, one way of finding an optimal solution to a 0-1 ILP is to simply enumerate all the 2^n assignments of 0s and 1s to the 0-1 variables. Each assignment is either infeasible (not satisfying all the constraints) or is feasible. Each feasible assignment gives a value to the objective function of the 0-1 (M)ILP. The feasible assignment that accords the largest value to the objective function constitutes an optimal solution to the 0-1 ILP.

A 0-1 MILP can be similarly solved. For each of the 2^n assignments, we solve the LP identical to the 0-1 MILP but in which the values of the 0-1 variables are set according to the assignment. Each of these 2^n either has a feasible solution or it does not. The LP with a feasible solution that gives the largest value to the objective function of the 0-1 MILP is an optimal solution to the 0-1 MILP.

If we choose to solve a 0-1 ILP or 0-1 MILP in this fashion, we are faced with an exponential (in the number of 0-1 variables of the program) number of evaluations. The BB method is designed to avoid a complete enumeration of the possible assignments of values to the 0-1 variables in finding an optimal solution to the program. The BB method in fact is usually able to finding an optimal solution by evaluating only a fraction of the possible assignments. We describe below how the BB method solves a 0-1 MILP. The discussion applies without change to the way it solves a 0-1 ILP as well.

The BB method is based on the concept of solving *relaxations* of the given 0-1 MILP. A **relaxation** of a 0-1 MILP is an LP in which each 0-1 variable of the 0-1 MILP is allowed to be continuous variable. Let $\mathbf{M}$ denote a (maximization) 0-1 MILP and $\mathbf{L}$ its relaxation. Let u_M denote the value of the objective function of $\mathbf{M}$ for an optimal solution to $\mathbf{M}$. Similarly, let u_L denote the value of the objective function of $\mathbf{L}$ for an optimal solution to $\mathbf{L}$. The main insight on which the BB method is based is that u_L *serves as an upper bound* to u_M. Thereby, first solving $\mathbf{L}$ and then progressively restituting the continuous variables in $\mathbf{L}$ that are 0-1 variables in $\mathbf{M}$ to their original 0-1 form, moves us progressively closer to a solution to $\mathbf{M}$.

The BB method is as follows.

Step 1: Solve $\mathbf{L}$. If every 0-1 variable receives a value of either 0 or 1 in the solution to $\mathbf{L}$, then terminate; an optimal solution to $\mathbf{M}$ has been found. If this is not the case (i.e., some 0-1 variables of $\mathbf{M}$ have received non-integer values in the solution), then go to Step 2.

Step 2: Select a 0-1 variable which has received a non-integer value in the solution to $\mathbf{L}$. This variable is called the **branching variable**. Let this variable be denoted by x. Add two relaxations of $\mathbf{M}$, denoted by $\mathbf{L}_x^0$ and $\mathbf{L}_x^1$ respectively, to the list of **active subproblems**. In $\mathbf{L}_x^0$, set x to 0 and in $\mathbf{L}_x^1$, set x to 1. That is, in $\mathbf{L}_x^0$ we add the constraint,

$$x = 0 \tag{4.89}$$

and in $\mathbf{L}_x^1$ we add the constraint,

$$x = 1 \tag{4.90}$$

Thus, in the two programs, x is effectively treated as a constant. We shall call any 0-1 variable whose value has been so fixed as a **constant 0-1 variable**.

Step 3: Select an LP from the list of active subproblems. Let this LP be denoted by $\mathbf{L}_a$. Denote the set of constant 0-1 variables of this LP by $\mathcal{X}(\mathbf{L}_a)$. For each $x' \in \mathcal{X}(\mathbf{L}_a)$, let $v(x')$ denote the value of x' in $\mathbf{L}_a$. Solve $\mathbf{L}_a$. Three cases are possible upon solving it:

(*i*) $\mathbf{L}_a$ does not have a feasible solution. In this case, drop $\mathbf{L}_a$ from the list of active subproblems.

(*ii*) $\mathbf{L}_a$ has a feasible solution and every 0-1 variable receives a value of either 0 or 1 in the solution. In this case, drop $\mathbf{L}_a$ from the list of active subproblems and add the solution to $\mathbf{L}_a$ to the list of **feasible solutions**.

(*iii*) $\mathbf{L}_a$ has a feasible solution but one or more 0-1 variables receive a non-integer value in the solution. In this case, select as the branching variable x a 0-1 variable *that has not been previous selected as the branching variable* and which has received a non-integer value in the solution to $\mathbf{L}_a$. Add the two LPs, $\mathbf{L}_x^0$ and $\mathbf{L}_x^1$, to the list of active subproblems, where x now denotes the new branching variable. $\mathbf{L}_x^0$ and $\mathbf{L}_x^1$ shall be called the **children** of $\mathbf{L}_a$. For each $i = 0, 1$, add the 0-1 constant variables from $\mathbf{L}_a$ to $\mathbf{L}_x^i$. That is, in $\mathbf{L}_x^i$, add the constraints,

$$x' = v(x'), \quad \forall\, x' \in \mathcal{X}(\mathbf{L}_a) \tag{4.91}$$

Thus, the children of $\mathbf{L}_a$ inherit its 0-1 constant variables.

Drop $\mathbf{L}_a$ from the list of active subproblems.

Step 4: If the list of active subproblems is empty, then terminate; the solution from the list of feasible solutions that gives the largest value of the objective function of **M** is the optimal solution to **M**. If the list is not empty, go to Step 3.

The BB method unfolds in a tree-like structure. At any given time, active subproblems represent the leaves of a tree and the branching on a variable represents the addition of two branches to a leaf. Notice that in Step 3, we are faced with making two choices: the choice of an active subproblem $\mathbf{L}_a$ and for Case (*iii*), the choice of a branching variable x. Judicious choices made in Step 3 result in a faster termination to the BB method. Two simple rules of thumb that are quite efficient in practice for making these choices are as follows. In choosing $\mathbf{L}_a$, choose the LP in the list whose solution gives the largest value of the objective function (note that here we are assuming that **M** has a maximization objective). In choosing x, choose a variable whose value is as close to 0.5 as possible.

4.4.3 Virtues Of Relaxation

We can thus appreciate that the time taken by the BB method to solve a 0-1 ILP or a 0-1 MILP is a function of the number of 0-1 variables in it. The larger the number of 0-1 variables, the wider is the choice for the BB method in selecting a branching variable, and the longer is the list of active subproblems. Therefore, if a 0-1 ILP is relaxed to an equivalent 0-1 MILP, then in theory, the BB method will take lesser time to solve the latter since the latter has fewer 0-1 variables than the former. So, **ILP1** takes longer to be solved than **MILP1** and similarly **ILP2** takes longer to be solved than **MILP2**.

Secondly, we may compare the two equivalent relaxations themselves. Given two 0-1 MILPs that are more or less identical, the one with the lesser number of 0-1 variables is potentially faster to solve by the BB method. **MILP1** and **MILP2** are identical except for one set of constraints. However, due to this difference, we can relax the z variables in the latter but not in the former, resulting in exponentially less 0-1 variables in it compared to the former. Thus, **MILP2** can be considered as an improvement in this regard.

There is yet another difference between the two 0-1 MILPs that we wish to point out. From the preceding discussion, it is clear that the BB method would take longer to solve **ILP2** than it does to solve **MILP2**. However, if we add a special instruction in the BB method, the solving times of the two programs are rendered identical. In order to describe this instruction, consider the following fact. In every solution (x^*, z^*) to **MILP2**, the following implication holds for every terminal joint history j,

$$x_i^*(j_i) = 1, \quad \forall i \in I \quad \Rightarrow \quad z^*(j) = 1 \tag{4.92}$$

This implication holds even though the z variables are not 0-1 variables. This means that irrespective of whether the z variables are 0-1 variables or not, the constraints of the program are such that this implication holds. Evidently, this implication also holds for **ILP2**. Thus, this means that when solving **ILP2**, if we add the instruction in the BB method that it should never select a z variable to branch on, then at each step, the BB method will limit itself to selecting only from the x variables corresponding to the terminal histories of the agents; in effect, it will be solving **MILP2**. Thus, with the aid of this instruction, as far as the BB method is concerned, the two programs are identical. Note that this instruction will not work in the case of **ILP1** since in that program the constraints are such that the above implication holds only if the z variables are 0-1 variables.

4.5 Summary

In this chapter, we have presented two 0-1 integer linear programs (ILPs), **ILP1** and **ILP2**, obtained by linearizing the NLP **NLP1** presented in Chapter 3. We have used two linearization techniques. **ILP2** can be considered as the better of the two by virtue of having far fewer constraints. The number of variables in both programs is exponential in T, the horizon of the problem and in n, the number of agents. The number of constraints in **ILP1** is exponential in T and in n, but the number of constraints in **MILP2** is exponential only in T.

We have also presented equivalent 0-1 MILP relaxations of the two 0-1 ILPs. These are respectively **MILP1** and **MILP2**. The latter is better of the two because it has far fewer 0-1 variables. The number of 0-1 variables in it is exponential in T and in n; the number of 0-1 variables in the former is exponential only in T.

The programs presented in this chapter were conceived using the property of finite horizon DEC-POMDPs that there in every finite horizon DEC-POMDP, there exists an optimal joint policy that is pure. In the next chapter, we present two more 0-1 MILPs based on a different property of finite horizon DEC-POMDPs.

Before moving on to the next chapter, we would like to point out the following perspective apropos the approach adopted in this chapter. As described in Chapter 2 (page 76), the value of an optimal joint policy for a DEC-POMDP is bound from above by the value of an optimal POMDP policy for the DEC-POMDP. The corollary of this is that an optimal DEC-POMDP joint policy constitutes a possibly *sub-optimal* POMDP policy. An optimal POMDP policy in the sequence form can be found by solving a linear program (LP). In Chapter 6, the definition of a POMDP policy in the sequence form (page 145) and the LP that finds an optimal policy (**LP4** page 145) is to be found. In the present context, by a 'POMDP policy', we mean a policy similar to an agent's policy in a DEC-POMDP, but which is defined over joint actions and joint

observations. It is essentially a centralized version of a joint policy (meaning that it is possible that the policy assigns different joint actions for different joint observations; this is not possible for a joint policy). In other words, a POMDP policy is meant to be used by the n agents together. Note that some, but not all POMDP policies can be decomposed into DEC-POMDP joint policies.

Now, finding an optimal POMDP policy is a problem of much lower complexity than the problem of finding an optimal DEC-POMDP joint policy. The former requires a linear program, the latter a 0-1 ILP or 0-1 MILP. So, suppose we find an optimal POMDP policy for the DEC-POMDP by solving an LP, denoted by **L**. The optimal policy π_p^* found upon solving **L** will not, in general, be an optimal DEC-POMDP joint policy. Only if π_P^* is decomposable into a DEC-POMDP joint policy, is it also an optimal DEC-POMDP joint policy. So, solving **L** does not guarantee us an optimal DEC-POMDP joint policy.

However, we can impose upon **L** certain structural constraints **C** that guarantee the decomposibility of π_P^* into an DEC-POMDP joint policy. Let **M** denote the mathematical program resulting upon adding **C** to **L**. Thus, an optimal solution to **M** will be an optimal DEC-POMDP joint policy, π_D^*. Seen another way, **M** finds a POMDP policy π_D^* that is a possibly sub-optimal POMDP policy, but which is decomposable into a DEC-POMDP joint policy. To be precise, π_D^* is a POMDP policy which has the largest value and which can be decomposed into a DEC-POMDP joint policy. This is the principle behind the approach adopted in this chapter. The two 0-1 MILPs **MILP1** and **MILP2** essentially represent **M**. The constraints **C** are the constraints (4.61)-(4.62) in **MILP1** and the constraints (4.71)-(4.72) in **MILP2**. The program that remains when these constraints are removed from the 0-1 MILP (and all 0-1 variables are converted to continuous variables) is **L**.

Chapter 5

An Optimal Nash Equilibrium Search Approach

5.1 Introduction

In this chapter, we shall develop more 0-1 mixed integer linear programs (MILPs) for finding an optimal T-period joint policy (in the sequence-form). These programs are based on the property that an optimal joint policy is *also* a Nash Equilibrium. Therefore, conditions that a joint policy is required to fulfill in order to be a Nash Equilibrium correspond to necessary conditions for a joint policy to be optimal. The necessary conditions for a joint policy to be a Nash Equilibrium are derived through the theorem of linear programming duality [Lue84]. They can also be derived through the Kuhn-Tucker Theorem [Fle87]. In comparison with the 0-1 MILPs presented in Chapter 4, the 0-1 MILPs presented in this chapter are smaller in size (less variables, less constraints).

These necessary conditions essentially require that the joint policy have *zero regret*. When a joint policy has zero regret, it means that it cannot be improved by changing the policy of only one agent in it. These conditions are not sufficient however. That is, while every optimal joint policy is certainly a Nash Equilibrium, not every Nash Equilibrium is an optimal joint policy. We can term an optimal joint policy as an *optimal Nash Equilibrium*. Since every optimal joint policy is also a Nash Equilibrium, it means that an optimal joint policy (Nash Equilibrium) must fulfill two agendas not one: (i) It must maximize value (ii) It must minimize regret (that is, bring it down to 0). While existing algorithms (and the ones we presented in Chapter 4) focus only on the first agenda to find an optimal joint policy, the algorithms we present in this chapter use both agendas to find an optimal joint policy.

In the 2-agent case, the necessary conditions form a mathematical program called a *linear complementarity problem* or LCP [Mur88]. When the number of agents is more than 2, the conditions form a *nonlinear* LCP (NLCP). Hence, for the 2-agent case, we convert an LCP to a 0- 1 MILP. For the three or more agents case, we present two different 0-1 MILPs, obtained by converting the NLCP. These 0-1 MILPs are both based on two properties of a DEC-POMDP: an optimal joint policy is also a Nash Equilibrium and an optimal joint policy can be pure. Thus, in conceiving them, we combine the properties considered in Chapter 4 and in this chapter.

5.2 Definitions And Preliminaries

A Nash Equilibrium is a joint policy in which each policy is a best response to the reduced joint policy formed by the other policies in the joint policy. A best response policy is defined as follows. A policy $p_i \in X_i$ of agent i is said to be a **best response** to an i-reduced joint policy $q_{-i} \in X_{-i}$ if there holds,

$$\mathcal{V}(\alpha, (p_i, q_{-i})) - \mathcal{V}(\alpha, (p'_i, q_{-i})) \quad \geq \quad 0, \quad \forall p'_i \in X_i \tag{5.1}$$

That is,

$$\sum_{h \in \mathcal{E}_i} \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k)\{p_i(h) - p'_i(h)\} \quad \geq \quad 0, \quad \forall p'_i \in X_i \tag{5.2}$$

Thereby, a Nash equilibrium is defined as follows. A joint policy $p \in X$ is a **Nash Equilibrium** if there holds,

$$\mathcal{V}(\alpha, p) - \mathcal{V}(\alpha, (p'_i, p_{-i}) \quad \geq \quad 0, \quad \forall i \in I \; \forall p'_i \in X_i \tag{5.3}$$

That is,

$$\sum_{h \in \mathcal{E}_i} \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} p_k(j'_k)\{p_i(h) - p'_i(h)\} \quad \geq \quad 0, \quad \forall i \in I, \forall p'_i \in X_i \tag{5.4}$$

Thus, if a joint policy is a Nash equilibrium, by changing only one policy in it, we will obtain a joint policy with a larger value than the (unchanged) joint policy. Since an optimal joint policy is defined as a joint policy whose value is as large or larger than any other joint policy, it is evident that an optimal joint policy is also a Nash equilibrium.

The necessary conditions for a joint policy to be a Nash equilibrium are essentially a set of *complementarity constraints*. It is interesting to consider what complementarity constraints imply without immediately going into much technical details. A complementarity constraint is a quadratic equation $xy = 0$ consisting of two linear terms x and y whose product is 0. Each complementarity constraint in the necessary conditions for a joint policy p to be a Nash equilibrium is of the form $p_i(h)\mu_i(h, p_{-i}) = 0$, for each agent $i \in I$ and for each history h of agent i. What are these quantities? $p_i(h)$ is of the weight of h in p_i. $\mu_i(h, p_{-i})$ is the *regret* of history h given the i-reduced joint policy p_{-i}. Informally, the regret of a history is the loss in the expected reward incurred by the agents if h is given zero weight. Therefore, the condition $p_i(h)\mu_i(h, p_{-i}) = 0$ means that in a joint policy that is a Nash equilibrium the regret of each history whose weight is nonzero in a policy of the joint policy, is 0. Therefore, the total regret of a joint policy (defined as the sum of the regrets of histories that are in the supports of the policies, that is those with nonzero weights in the policies, of the joint policy) is 0.

The definition of the regret of a history requires the concept of an information set. An **information set** ι of agent i is a sequence of even length in which the elements in odd positions are actions of the agent (members of A_i) and those in even positions are observations of the agent (members of O_i). Thus, despite its name, an information set is not a set but a sequence. An information set is so called because given an information set ι we can use it to group all possible joint histories that can occur from the agent's perspective given ι. In other words, ι circumscribes the agent's knowledge about which joint history may occur at the end of that

period. The number of actions in an information set shall be called its **length**. An information set of length $t \geq 0$ has t actions and t observations. An information set of length 0 shall be called the **null information set**, denoted by $\varnothing$. An information set of length T - 1 shall be called a **terminal information set**. Information sets of lengths less than T - 1 shall be called *nonterminal information sets*. The information set of a history h, denoted by $\iota(h)$, is the information set obtained by dropping the last action in the history. Thus, history h is said to belong to information set $\iota(h)$. The set of information sets of length t of agent i shall be denoted by $\mathcal{I}_i^t$. The set of information sets of lengths less than or equal to T - 1 shall be denoted by $\mathcal{I}_i$. Note that the size of $\mathcal{I}_i$ is $\sum_{t=1}^{T} |A_i|^{t-1}|O_i|^{t-1}$. It is exponential in T. The size of $\mathcal{I}_i$ shall be denoted by m_i.

The regret of a history is related to the *value* of the information set to which it belongs. Both terms are *relative* terms and not absolute ones. They are a function of the reduced joint policy formed by the policies of the other agents. The value of an information set ι of agent i given an i-reduced joint policy q_{-i} is the maximum value obtainable if the following two conditions are satisfied:

(1) The other agents choose actions according to q_{-i} for all the T periods.

(2) If agent i reaches ι (using any policy), he takes only optimal actions from that point onwards. "Reaching" an information set means that the sequence of actions taken by the agent and observations received by the agent are according to ι.

The value of an information set is defined as follows. The **value** $\lambda_i^*(\iota, q_{-i})$ of information set $\iota \in \mathcal{I}_i$ of agent i for a given i-reduced joint policy $q_{-i} = (q_1, q_2, \ldots, q_n) \in P_{-i}$ is defined as follows.

- If ι is a terminal information set then,

$$\lambda_i^*(\iota, q_{-i}) = \max_{h \in \iota} \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \tag{5.5}$$

- if ι is a nonterminal information set then,

$$\lambda_i^*(\iota, q_{-i}) = \max_{h \in \iota} \sum_{o \in O_i} \lambda_i^*(ho, q_{-i}) \tag{5.6}$$

  where ho denotes the information set obtained on concatenating o to h.

The history selected by the max operator shall be called the **optimal history** of ι.

Thereby, the regret of a history h of agent i given an i-reduced joint policy q_{-i} measures the difference in the expected reward when:

(1) The other agents choose actions according to q_{-i} for all the T periods.

(2) If agent i reaches $\iota(h)$ (using any policy), he takes the last action of h instead of the last action of the optimal history of $\iota(h)$. (Recall that $\iota(h)$ denotes the information set to which h belongs).

In order to formally define the regret of a history, we shall define the contribution of a history. The **contribution of a history** h of agent i given an i-reduced joint policy $q_{-i} = (q_1, q_2, \ldots, q_n) \in P_{-i}$ is defined as follows.

- If h is a terminal history then its contribution is the quantity,

$$\sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \tag{5.7}$$

- If h is a nonterminal history then its contribution is defined to be 0.

Then, the regret of a history is defined as follows. The **regret** $\mu_i(h, q_{-i})$ of a history h of agent i given an i-reduced joint policy $q_{-i} = (q_1, q_2, \ldots, q_n) \in P_{-i}$ is defined as follows.

- If h is a terminal history then,

$$\mu_i(h, q_{-i}) \quad = \quad \lambda_i^*(\iota(h), q_{-i}) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \tag{5.8}$$

- If h is a nonterminal history then,

$$\mu_i(h, q_{-i}) \quad = \quad \lambda_i^*(\iota(h), q_{-i}) - \sum_{o \in O_i} \lambda_i^*(ho, q_{-i}) \tag{5.9}$$

Thus, the regret of a terminal history is the difference between the value of the information set to which it belongs and the contribution of the history. The regret of a nonterminal history is the difference between the value of the information set to which it belongs and the values of the information sets to which the children of the history belong. Notice that by definition, the regret of a history cannot be a negative number. It is either 0 or greater than 0.

Figure (5.1) illustrates the relationship between values of information sets and the regrets of histories. The tree shows the possible histories of agent i that can occur for two periods ($T = 2$). A_i is assumed to be $\{c, d\}$ and O_i is assumed to be $\{u, v\}$. Thus, the set of information sets of agent i is $\mathcal{I}_i = \{\varnothing, cu, cv, du, dv\}$. The number at each leaf is the contribution of the terminal history that ends in that leaf for a given i-reduced joint policy $q_{-i} = (q_1, q_2, \ldots, q_n)$. For instance the 2 at the end of the history cuc is the quantity $\sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (cuc, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k)$. The value of information set cu is 2 because if the agent reaches this information set, he can take action c and the agents together can claim an expected reward of 2. Similarly, the value of information set cv is 1. If the agent reaches this information set, he can take c or d; both actions yield an expected reward of 1. The values of information sets du and dv are respectively 4 and 1. The value of $\varnothing$ is the greater of the following two sums: $(\lambda_i^*(cu, q_{-i}) + \lambda_i^*(cv, q_{-i}))$ and $(\lambda_i^*(du, q_{-i}) + \lambda_i^*(dv, q_{-i}))$. These sums are respectively 3 and 5. Hence, $\lambda_i^*(\varnothing, q_{-i}) = 5$. The lines in bold show histories whose regret is 0. For instance, the regret of cuc is 2 - 2 = 0 while the regret of cuv is 2 - 1 = 1. Similarly, the regret of duc is 4 - 3 = 1 while the regret of dud is 4 - 4 = 0. The regret of c is 5 - 3 = 2 while the regret of d is 5 - 5 = 0.

5.2.1 Linear Programming Duality

In deriving necessary conditions for a joint policy to be a Nash equilibrium, we shall use the theorem of linear programming duality. Every linear program (LP) has a converse linear program called its dual. The first LP is called the primal to distinguish it from its dual. If the primal maximizes a quantity, the dual minimized the quantity. If there are n' variables and m'

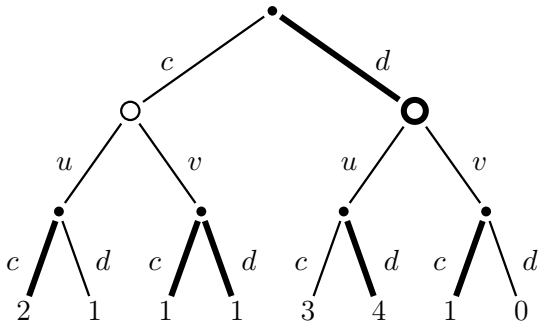


Figure 5.1: Relationship Between Values Of Information Sets And Regrets Of Histories.

constraints in the primal, there are m' variables and n' constraints in the dual. Consider the following (primal) LP.

$$\text{Maximize} \quad \sum_{i=1}^{n'} c(i)x(i) \tag{5.10}$$

Subject To:

$$\sum_{i=1}^{n'} a(i,j)x(i) \quad = \quad b(j), \quad j = 1, 2, \ldots, m' \tag{5.11}$$

$$x(i) \quad \geq \quad 0, \quad i = 1, 2, \ldots, n' \tag{5.12}$$

The LP has one variable $x(i)$ for each $i = 1$ to n'. The data of the LP consists of numbers $c(i)$ for each $i = 1$ to n', the numbers $b(j)$ for each $j = 1$ to m' and the numbers $a(i,j)$ for each $i = 1$ to n' and for each $j = 1$ to m'. The LP thus has n' variables and m' constraints. The dual of this LP is the following LP.

$$\text{Minimize} \quad \sum_{j=1}^{m'} b(j)y(j) \tag{5.13}$$

Subject To:

$$\sum_{j=1}^{m'} a(i,j)y(j) \quad \geq \quad c(i), \quad i = 1, 2, \ldots, n' \tag{5.14}$$

$$y(j) \quad \in \quad [-\infty, +\infty], \quad j = 1, 2, \ldots, m' \tag{5.15}$$

The dual LP has one variable $y(j)$ for each $j = 1$ to m'. Each $y(j)$ variable is a **free** variable. That is, it is allowed to take any positive or negative value in $\mathbb{R}$ or to equal 0. The dual LP has m' variables and n' constraints.

The theorem of linear programming duality is as follows.

Theorem 5.1. *([Lue84]) If either a primal LP or its dual LP has a finite optimal solution, then so does the other, and the corresponding values of the objective functions are equal.*

Applying this theorem to the primal-dual pair given above, there holds,

$$\sum_{i=1}^{n'} c(i)x^*(i) \quad = \quad \sum_{j=1}^{m'} b(j)y^*(j) \tag{5.16}$$

where x^* denotes an optimal solution to the primal and y^* denotes an optimal solution to the dual.

5.3 Necessary Conditions For A Best Response Policy

The derivation of the necessary conditions for a Nash equilibrium consists of deriving the necessary conditions for a policy to be a best response to a reduced joint policy. A policy of agent i that is a best response to an i-reduced joint policy can be determined through a (primal) linear program. The following primal LP finds a policy of agent i that is a best response to an i-reduced joint policy $q_{-i} \in X_{-i}$.

$$\text{Maximize} \quad \sum_{h \in \mathcal{E}_i} \{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \} x_i(h) \tag{5.17}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) \quad = \quad 1 \tag{5.18}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) \quad = \quad 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{5.19}$$

$$x_i(h) \quad \geq \quad 0, \quad \forall h \in \mathcal{H}_i \tag{5.20}$$

The LP has one variable in $x_i(h)$ for each history $h \in \mathcal{H}_i$ representing the weight of h. The LP has one constraint per information set of agent i. In other words, each constraint of the LP is uniquely labeled by an information set. For instance, the constraint (5.18) is labeled the null information set $\varnothing$, and for each nonterminal history h and for each observation o, the corresponding constraint in (5.19) is labeled by the information set ho. Thus, the LP has n_i variables and m_i constraints. A solution x_i^* to this LP is a best response policy to q_{-i}.

The dual of this LP is the following LP.

$$\text{Minimize} \quad y_i(\varnothing) \tag{5.21}$$

Subject To:

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) \quad \geq \quad 0, \quad \forall h \in \mathcal{N}_i \tag{5.22}$$

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \quad \geq \quad 0, \quad \forall h \in \mathcal{E}_i \tag{5.23}$$

$$y_i(\iota) \quad \in \quad [-\infty, +\infty], \quad \forall \iota \in \mathcal{I}_i \tag{5.24}$$

where $\iota(h)$ denotes the information set to which h belongs. The dual has one free variable $y_i(\iota)$ for every information set of agent i. It has one constraint per history of the agent. Thus, the

dual has m_i variables and n_i constraints. Note that the objective of the dual is to minimize only $y_i(\varnothing)$ because in the primal LP, the right-hand side of all the constraints except the very first one, is a 0. If we consider the definition of the value of an information set, we can see that the dual finds the value of every information set of agent i given q_{-i}. That is, for each $\iota \in \mathcal{I}_i$, $y_i^*(\iota)$ is the value of information set ι given $(q_1, q_2, \ldots, q_n)$.

The working of the dual might not be evident. How does minimizing $y_i(\varnothing)$ enable us to find the values of all information sets (including $\varnothing$)? In solving the dual LP we attempt to find the smallest number that can be accorded to $y_i(\varnothing)$ such that the constraints are satisfied. That this smallest number also happens to be the value of the null information set is best explained by the following example. Consider again the tree shown in Figure (5.1). If the dual is set up for the example given in the figure, then its constraints are as follows:

$$\begin{aligned}
y_i(\varnothing) - y_i(cu) - y_i(cv) &\geq 0 \\
y_i(\varnothing) - y_i(du) - y_i(dv) &\geq 0 \\
y_i(cu) &\geq \text{Cont}(cuc) \\
y_i(cu) &\geq \text{Cont}(cud) \\
y_i(cv) &\geq \text{Cont}(cvc) \\
y_i(cv) &\geq \text{Cont}(cvd) \\
y_i(du) &\geq \text{Cont}(duc) \\
y_i(du) &\geq \text{Cont}(dud) \\
y_i(dv) &\geq \text{Cont}(dvc) \\
y_i(dv) &\geq \text{Cont}(dvd)
\end{aligned}$$

where $\text{Cont}(h)$ denotes the contribution $\sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k)$ of h. In the example, the contributions of the histories are as follows.

$$\begin{aligned}
\text{Cont}(cuc) = 2, \quad & \text{Cont}(cud) = 1 \\
\text{Cont}(cvc) = 1 \quad & \text{Cont}(cvd) = 1 \\
\text{Cont}(duc) = 3 \quad & \text{Cont}(dud) = 4 \\
\text{Cont}(dvc) = 1 \quad & \text{Cont}(dvd) = 0
\end{aligned}$$

Due to the constraints, $y_i(cu)$ will assume a value of at least 2, $y_i(cv)$ will assume a value of at least 1, $y_i(du)$ will assume a value of at least 4 and $y_i(dv)$ will assume a value of at least 1. Therefore, we will have,

$$\begin{aligned}
y_i(\varnothing) - 3 &\geq 0 \\
y_i(\varnothing) - 5 &\geq 0
\end{aligned}$$

Therefore, $y_i(\varnothing)$ will assume a value of at least 5. Now, there are an infinite number of values that $y_i(\varnothing)$ can assume. But since we are minimizing $y_i(\varnothing)$, it will assume a value of *exactly* 5. This will cause the other variables to also assume the values of their respective lower bounds. Thus, by minimizing $y_i(\varnothing)$, we find the maximum expected rewards of all the information sets of the agent.

The dual also indirectly finds the regrets of the histories of agent i given q_{-i} since they can be deduced from the values of the information sets. But we can also use variables in the dual to explicitly find the regrets of histories. These variables play the role of "surplus" variables in making the inequalities in the dual to equalities. The transformed dual with these surplus variables is as follows:

$$\text{Minimize} \quad y_i(\varnothing) \tag{5.25}$$

Subject To:

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) = w_i(h), \quad \forall h \in \mathcal{N}_i \tag{5.26}$$

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) = w_i(h), \quad \forall h \in \mathcal{E}_i \tag{5.27}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad \forall \iota \in \mathcal{I}_i \tag{5.28}$$

$$w_i(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{5.29}$$

As can be seen, the transformed dual is identical to the dual except that we have now used a variable $w_i(h)$ for each history h of agent i; $w_i(h)$ represents the regret of history h. A solution of the transformed dual consists of values of information sets and regrets of histories. That is, for each $\iota \in \mathcal{I}_i$, $y_i^*(\iota)$ is the value of information set ι given q_{-i} and for each $h \in \mathcal{H}_i$, $w_i^*(h)$ is the regret of history h given q_{-i}.

Applying the theorem of LP duality to the primal LP (5.17)-(5.20) and the transformed dual LP (5.25)-(5.29), we obtain

$$\sum_{h \in \mathcal{E}_i} \Big\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \Big\} x_i^*(h) = y_i^*(\varnothing) \tag{5.30}$$

Thus, the value of the joint policy (x_i^*, q_{-i}) can be expressed either as,

$$\mathcal{V}(\alpha, (x_i^*, q_{-i})) = \sum_{h \in \mathcal{E}_i} \Big\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \Big\} x_i^*(h) \tag{5.31}$$

or as,

$$\mathcal{V}(\alpha, (x_i^*, q_{-i})) = y_i^*(\varnothing) \tag{5.32}$$

Due to the constraints (5.18) and (5.19) of the primal LP, there holds,

$$y_i^*(\varnothing) = y_i^*(\varnothing) \Big\{ \sum_{a \in A_i} x_i^*(a) \Big\} + \sum_{h \in \mathcal{N}_i} \sum_{o \in O_i} y_i^*(ho) \Big\{ - x_i^*(h) + \sum_{a \in A_i} x_i^*(hoa) \Big\} \tag{5.33}$$

Due to (5.18), the first term in the braces is 1 and due to (5.20) each of the remaining terms in braces is 0. The right hand side of (5.33) can be rewritten as,

$$\sum_{a \in A_i} x_i^*(a) \Big\{ y_i^*(\varnothing) - \sum_{o \in O_i} y_i^*(ao) \Big\} + \sum_{h \in \mathcal{N}_i \setminus A_i} x_i^*(h) \Big\{ y_i^*(\iota(h)) - \sum_{o \in O_i} y_i^*(ho) \Big\} + \sum_{h \in \mathcal{E}_i} x_i^*(h) y_i^*(\iota(h)) =$$
$$\sum_{h \in \mathcal{N}_i} x_i^*(h) \Big\{ y_i^*(\iota(h)) - \sum_{o \in O_i} y_i^*(ho) \Big\} + \sum_{h \in \mathcal{E}_i} x_i^*(h) y_i^*(\iota(h)) \tag{5.34}$$

Further, from (5.30) we have,

$$\sum_{h\in\mathcal{E}_i}\{\sum_{j'\in\mathcal{E}_{-i}}\mathcal{R}(\alpha,(h,j'))\prod_{k\in I\setminus\{i\}}q_k(j'_k)\}x_i^*(h) \;=\; \sum_{h\in\mathcal{N}_i}x_i^*(h)\{y_i^*(\iota(h))-\sum_{o\in O_i}y_i^*(ho)\}+\sum_{h\in\mathcal{E}_i}x_i^*(h)y_i^*(\iota(h)) \tag{5.35}$$

Therefore,

$$\sum_{h\in\mathcal{N}_i}x_i^*(h)\{y_i^*(\iota(h))-\sum_{o\in O_i}y_i^*(ho)\}+\sum_{h\in\mathcal{E}_i}x_i^*(h)\{y_i^*(\iota(h))-\sum_{j'\in\mathcal{E}_{-i}}\mathcal{R}(\alpha,(h,j'))\prod_{k\in I\setminus\{i\}}q_k(j'_k)\} \;=\; 0 \tag{5.36}$$

Due to the constraints (5.26) and (5.27) of the transformed dual, we can furthermore rewrite (5.36) as simply,

$$\sum_{h\in\mathcal{N}_i}x_i^*(h)w_i^*(h)+\sum_{h\in\mathcal{E}_i}x_i^*(h)w_i^*(h) \;=\; 0 \tag{5.37}$$

$$\sum_{h\in\mathcal{H}_i}x_i^*(h)w_i^*(h) \;=\; 0 \tag{5.38}$$

Now, (5.38) is a sum of n_i products, n_i being the size of $\mathcal{H}_i$. Each product in this sum is necessarily 0 because both $x_i(h)$ and $w_i(h)$ are constrained to be nonnegative in the primal and the dual respectively. Hence, (5.38) is equivalent to,

$$x_i^*(h)w_i^*(h) \;=\; 0, \quad \forall h\in\mathcal{H}_i \tag{5.39}$$

Each equation $x_i^*(h)w_i^*(h)$ is called a **complementarity constraint** or an **equilibrium constraint**. Each complementarity constraint implies that either the weight of h be zero or its regret be 0. Both cannot be false at the same time.

Therefore, we can conclude the following that the necessary conditions for a policy $p_i \in P_i$ of agent i to be a best response policy to a given i-reduced joint policy $q_{-i} \in X_{-i}$ are that,

$$p_i(h)\mu_i(h,q_{-i}) \;=\; 0, \quad \forall h\in\mathcal{H}_i \tag{5.40}$$

5.4 Necessary Conditions For A Nash Equilibrium

The necessary conditions for joint policy to be a Nash Equilibrium are obtained by generalizing the reasoning of obtaining the necessary conditions for a best response policy. That is, we must assume that each agent *simultaneously* with the other agents attempts to find a policy that is a best response to the reduced joint policies formed by the other agents' policies. This implies that we must set up and solve simultaneously n pairs of primal-dual linear programs. The generalization yields the required necessary conditions. The necessary conditions for a joint policy $p \in X$ to be a Nash Equilibrium are that,

$$p_i(h)\mu_i(h,p_{-i}) \;=\; 0, \quad \forall i\in I, \forall h\in\mathcal{H}_i \tag{5.41}$$

Thus, as stated earlier, the necessary conditions consist of a set of complementarity constraints per agent. If we define the regret of a policy, given a reduced joint policy formed by the policies of other agents, as the sum of the regrets of the histories that are in the support of the policy. Then the complementarity constraints imply that if $(p_1, p_2, \ldots, p_n)$ is a Nash equilibrium, the regret of every policy p_i is 0. Now, the regret of a history given a reduced joint policy is a nonnegative quantity. Therefore, the regret of a policy given a reduced joint policy is also a nonnegative quantity. Thus, the smallest regret achievable by a policy given a reduced joint policy is 0. And in a Nash equilibrium, every policy achieves a regret of 0. In other words, if the regret of a policy given a reduced joint policy, is greater than 0, then it simply means that that policy is not a best response to the reduced joint policy. If we define the regret of a joint policy as the sum of the regrets of the policies in it, then a Nash equilibrium achieves 0 regret. Thus, we as stated in the introduction, can reconsider what optimality means in numerical terms in a DEC-POMDP. On the one hand, the maximum value in a DEC-POMDP is a number that depends on the reward function, the state transition probabilities and the observation probabilities, and the initial state. This number is unknown to us, unless we engage in computing it. On the other hand, the minimum regret in a DEC-POMDP is a known constant: it is always zero. We can now appreciate that an optimal joint policy is required to produce maximum value and minimum regret.

Note that when these n pairs of primals and duals are solved to obtain the necessary conditions, there holds for each agent $i \in I$,

$$\sum_{h \in \mathcal{E}_i} \Big\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) \Big\} x_i^*(h) \quad = \quad y_i^*(\varnothing) \tag{5.42}$$

and

$$x_i^*(h) w_i^*(h) \quad = \quad 0, \quad \forall h \in \mathcal{H}_i \tag{5.43}$$

The value of the joint policy $x^* = (x_1^*, x_2^*, \ldots, \ldots, x_n^*)$ can be expressed either as, for each agent $i \in I$,

$$\mathcal{V}(\alpha, x^*) \quad = \quad \sum_{h \in \mathcal{E}_i} \Big\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) \Big\} x_i^*(h) \tag{5.44}$$

or as,

$$\mathcal{V}(\alpha, x^*) \quad = \quad y_i^*(\varnothing) \tag{5.45}$$

Thus, $\forall i, j \in I$,

$$y_i^*(\varnothing) \quad = \quad y_j^*(\varnothing) \tag{5.46}$$

5.4.1 Nonlinear Program To Find Optimal Joint Policy

The necessary conditions obtained above can be transformed to the following nonlinear program (NLP) for finding an optimal n-agent joint policy.

$$\text{Maximize} \quad y_1(\varnothing) \tag{5.47}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{5.48}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{5.49}$$

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{5.50}$$

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} x_k(j'_k) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.51}$$

$$x_i(h) w_i(h) = 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.52}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.53}$$

$$w_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.54}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad \forall i \in I, \forall \iota \in \mathcal{I}_i \tag{5.55}$$

We shall henceforth refer to the NLP (5.47)- (5.55) as **NLP2**.

Given a global maximum point (x^*, y^*, w^*) of this NLP, $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an optimal joint policy. Note that $y_1(\varnothing)$ can be replaced by $y_i(\varnothing)$ for any $i \in I$. When $n = 2$, the constraints of **NLP2** constitute a linear complementarity problem (LCP), and when $n > 2$, they constitute a nonlinear complementarity problem (NLCP). For $n = 2$, the LCP we obtain is identical to the one obtained by the KMvS approach [KMvS94]. Note that when $n = 2$, **NLP2** is also called a **complementary problem** [Iba71] or a linear program with equilibrium constraints (LPEC).

When $n = 2$, every constraint in this NLP is a linear constraint with the exception of the complementarity constraints, since they remain quadratic even in that case. When $n > 2$, every constraint in **NLP2** is a linear constraint with the exception of the complementarity constraints and the information sets values constraints (5.51) since they contain the nonlinear term $\prod_{k \in I \setminus \{i\}} x_k(j'_k)$. Due to the presence of nonlinear constraints in **NLP2** for both $n = 2$ and $n > 2$, we are not guaranteed to find a global maximum point of the program by solving the program directly. The program must be linearized. When $n = 2$, only the complementarity constraints must be linearized. When $n > 2$, the complementarity constraints as well as the nonlinear term appearing in (5.51) must be linearized. The linearization of the complementarity constraints is the same be $n = 2$ or $n > 2$. Therefore, in the next section, we shall see how each complementarity constraint can be separated into a pair of linear constraints with the aid of a 0-1 variable.

5.5 Linearization Of Complementarity Constraints

Consider a complementarity constraint $ab = 0$ in variables a and b. Assume that the lower bound on the values of a and b is 0. Let the upper bounds on the values of a and b be respectively u_a and u_b. Now let c be a 0-1 variable. That is, c is allowed to assume a value of either 0 or 1. Then, the complementarity constraint $ab = 0$ can be separated into the following equivalent pair of linear constraints,

$$a \leq u_a c \tag{5.56}$$

$$b \leq u_b (1 - c) \tag{5.57}$$

In other words, if this pair of constraints is satisfied, then it is surely the case that $ab = 0$. This is easily verified. c can either be 0 or 1. If $c = 0$, then a will be set to 0 because a is constrained to be not more than $u_a c$ (and not less than 0); if $c = 1$, then b will be set to 0 since b is constrained to be not more than $u_b(1 - c)$ (and not less than 0). In either case, $ab = 0$.

Now consider each complementarity constraint $x_i(h)w_i(h) = 0$ from **NLP2**. We wish to separate this constraint into a pair of linear constraints. We recall that $x_i(h)$ represents the weight of h and $w_i(h)$ represents the regret of h. The first requirement to convert this constraint to a pair of linear constraints is that the lower bound on the values of the two terms be 0. This is indeed the case since $x_i(h)$ and $w_i(h)$ are both constrained to be nonnegative in the NLP. Next, we require upper bounds on the weights of histories and regrets of histories. We have shown in Chapter 4 that the upper bound on the value of $x_i(h)$ for each h is 1. For the upper bounds on the regrets of histories, we require some notation. An upper bound of the regret of a history of an agent is the maximum value the regret of the history can assume regardless of the reduced joint policy formed by the policies of the other agents. We denote the upper bound on the regret of a history h of agent i by $\mathcal{U}_i(h)$. The definitions of upper bounds on the regrets of histories are given in the next subsection. Given an upper bound $\mathcal{U}_i(h)$ on the regret of a history h of agent i, the complementarity constraint $x_i(h)w_i(h) = 0$ can separated into a pair of linear constraints by using a 0-1 variable $b_i(h)$ as follows,

$$\begin{aligned} x_i(h) &\leq 1 - b_i(h) && (5.58)\\ w_i &\leq \mathcal{U}_i(h)b_i(h) && (5.59)\\ b_i(h) &\in \{0, 1\} && (5.60) \end{aligned}$$

5.5.1 Upper Bounds On Regrets

In any policy p_i of agent i there holds,

$$\sum_{h \in \mathcal{E}_i} p_i(h) = |O_i|^{T-1} \quad (5.61)$$

Therefore, in every i-reduced joint $(q_1, q_2, \ldots, q_n) \in X_{-i}$, there holds,

$$\sum_{j' \in \mathcal{E}_{-i}} \prod_{k \in I \setminus \{i\}} q_k(j'_k) = \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} \quad (5.62)$$

Since the regret of a terminal history h of agent i given $(q_1, q_2, \ldots, q_n)$ is defined as,

$$\mu_i(h, q) = \max_{h' \in \iota(h)} \sum_{j' \in \mathcal{E}_{-i}} \prod_{k \in I \setminus \{i\}} q_k(j'_k)\big\{\mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j'))\big\} \quad (5.63)$$

we can conclude that the **upper bound** $\mathcal{U}_i(h)$ on the regret of a **terminal history** $h \in \mathcal{E}_i$ of agent i is,

$$\mathcal{U}_i(h) = \prod_{k \in I \setminus \{i\}} |O_k|^{T-1}\{\max_{h' \in \iota(h)} \max_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h', j')) - \min_{j'' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j''))\} \quad (5.64)$$

Now let us consider the upper bounds on the regrets of nonterminal histories. Let ι be an information set of length t of agent i. Let $\mathcal{E}_i(\iota) \subseteq \mathcal{E}_i$ denote the set of terminal histories of agent i such the first $2t$ elements of each history in the set are identical to ι. Let h be a history of length $t \leq T$ of agent i. Let $\mathcal{E}_i(h) \subseteq \mathcal{E}_i$ denote the set of terminal histories such that the first $2t$ - 1 elements of each history in the set are identical to h. Since in any policy p_i of agent i, there holds,

$$\sum_{h' \in \mathcal{E}_i(h)} p_i(h') \leq |O_i|^{T-t} \tag{5.65}$$

we can conclude that the **upper bound** $\mathcal{U}_i(h)$ on the regret of a **nonterminal history** $h \in \mathcal{N}_i$ of length t agent i is,

$$\mathcal{U}_i(h) = L_i\{\max_{h' \in \mathcal{E}_i(\iota(h))} \max_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h', j')) - \min_{g \in \mathcal{E}_i(h)} \min_{j'' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (g, j''))\} \tag{5.66}$$

where,

$$L_i \;=\; |O_i|^{T-t} \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} \tag{5.67}$$

Notice that if $t = T$ (that is, h is terminal) (5.66) reduces to (5.64).

5.6 0-1 Mixed Integer Linear Program: Two Agent Case

NLP2 can be transformed to the following 0-1 mixed integer linear program (MILP) by converting each complementarity constraint in it to a pair of equivalent linear constraints and by using 0-1 variables as described in the previous section:

$$\text{Maximize} \quad y_1(\varnothing) \tag{5.68}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{5.69}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad i = 1, 2, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{5.70}$$

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) = w_i(h), \quad i = 1, 2, \forall h \in \mathcal{N}_i \tag{5.71}$$

$$y_1(\iota(h)) - \sum_{h \in \mathcal{E}_2} \mathcal{R}(\alpha, (h, h'))x_2(h') = w_1(h), \quad \forall h \in \mathcal{E}_1 \tag{5.72}$$

$$y_2(\iota(h)) - \sum_{h \in \mathcal{E}_1} \mathcal{R}(\alpha, (h', h))x_1(h') = w_2(h), \quad \forall h \in \mathcal{E}_2 \tag{5.73}$$

$$x_i(h) \leq 1 - b_i(h), \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{5.74}$$

$$w_i(h) \leq \mathcal{U}_i(h)b_i(h), \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{5.75}$$

$$x_i(h) \geq 0, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{5.76}$$

$$w_i(h) \geq 0, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{5.77}$$

$$b_i(h) \in \{0, 1\}, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{5.78}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad i = 1, 2, \forall \iota \in \mathcal{I}_i \tag{5.79}$$

We shall henceforth refer to the 0-1 MILP (5.68)-(5.79) as **MILP3**. Note that $\mathcal{U}_i(h)$ denotes the upper bound on history h.

The variables of the program are the vectors x_i, w_i, b_i and y_i for each agent i. Note that for each agent $i \in I$ and for each history h of agent i, $\mathcal{U}_i(h)$ denotes the upper bound on the regret of history h.

A solution (x^*, y^*, w^*, b^*) to **MILP3** consists of the following quantities. (i) An optimal joint policy $x^* = (x_1^*, x_2^*)$ which may be a stochastic. (ii) For each agent $i = 1, 2$, for each history $h \in \mathcal{H}_i$, $w_i^*(h)$, the regret of h given the policy x_{-i}^* of the other agent. (iii) For each agent $i = 1, 2$, for each information set $\iota \in \mathcal{I}_i$, $y_i^*(\iota)$, the value of ι given the policy x_{-i} of the other agent. (iv) For each agent $i = 1, 2$, the vector b_i^* simply tells us which histories are not in the support of x_i^*; each history h of agent i such that $b_i^*(h) = 1$ is *not* in the support of x_i^*. Note that we can replace $y_1(\varnothing)$ by $y_2(\varnothing)$ in the objective function without affecting the program. We have the following result.

Theorem 5.2. *Given a solution (x^*, w^*, y^*, b^*) to* **MILP3**, *$x^* = (x_1^*, x_2^*)$ is an optimal joint policy*

Proof: Due to the policy constraints of each agent, each x_i^* is a policy of agent i. Due to the constraints (5.71)-(5.73), y_i^* contains the values of the information sets of agent i given x_{-i}^*. Due to the complementarity constraints (5.74)-(5.75), each x_i^* is a best response to x_{-i}^*. Thus (x_1^*, x_2^*) is a Nash equilibrium. Finally, by maximizing the value of the null information set of agent 1, we are effectively maximizing the value of (x_1^*, x_2^*). Thus (x_1^*, x_2^*) is an optimal joint policy.*Q.E.D*

In comparison with **MILP1** and **MILP2** presented in Chapter 4, **MILP3** constitutes a particularly effective program for finding a 2-agent optimal T-period joint policy because it is a much smaller program compared to them. While the number of variables required by those programs is exponential in T and in n, the number of variables required by **MILP3** is exponential only in T. This represents a major reduction in size (and as the computational experience shows, in time). In fact, **MILP3** constitutes the smallest program presented in this thesis for solving 2-agent finite horizon DEC-POMDPs.

5.7 0-1 Mixed Integer Linear Program: Three Or More Agents Case

When the number of agents is more than 2, **NLP2** cannot be transformed to 0-1 MILP by merely replacing each complementarity constraint by a pair of equivalent linear constraints and by using 0-1 variables. If we wish to convert the NLP to a 0-1 MILP, we must also convert the nonlinear term $\prod_{k \in I \setminus \{i\}} x_k(j'_k)$ in (5.51) to an equivalent linear term. This term can be linearized as follows. In linearizing the term we rely on the fact that in a DEC-POMDP there always exists an optimal joint policy that is pure. Hence, the resultant 0-1 MILP shall be capable of finding only a pure optimal joint policy, and not a stochastic optimal joint policy as was possible when $n = 2$.

For the linearization, the changes/additions to **NLP2** are as follows.

(C1) For each agent $i \in I$ and for each terminal history $h \in \mathcal{E}_i$, we constrain the variable $x_i(h)$

to be a 0-1 variable. That is, we replace (5.53) by,

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{5.80}$$
$$x_i(h) \in \{0,1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.81}$$

(C2) For every joint terminal history $j \in \mathcal{E}$, we introduce a variable $z(j)$ which we constrain to be in the interval [0, 1]. That is, we add,

$$z(j) \in [0,1], \quad \forall j \in \mathcal{E} \tag{5.82}$$

(C3) For each agent $i \in I$, we add the following set of constraints,

$$\sum_{j' \in \mathcal{E}_{-i}} z(h,j') = \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall h \in \mathcal{E}_i \tag{5.83}$$

(C4) We add the following constraint,

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{5.84}$$

(C5) We replace the nonlinear equations (5.51) by the linear equations

$$y_i(\iota(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, (h, j_{-i})) z(j) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.85}$$

The effects of these changes/additions are as follows. Due to (C1), every solution x^* to the resultant program is a pure joint policy; each x_i^* is a pure policy of agent i. (C2), (C3) and (C4) ensure that the following double implication holds for every terminal joint history $j \in \mathcal{E}$,

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{5.86}$$

The motivation behind (C5) is explained as follows. Due to (C1), for each agent i, x_i^* is a pure policy and so, for each $h \in \mathcal{H}_i$, $x_i^*(h)$ is either 0 or 1. Moreover, there holds,

$$\sum_{h' \in \mathcal{E}_i} x_i^*(h) = |O_i|^{T-1} \tag{5.87}$$
$$\frac{\sum_{h' \in \mathcal{E}_i} x_i^*(h)}{|O_i|^{T-1}} = 1 \tag{5.88}$$

Therefore, due to (C2)-(C4), in any solution to the changed program, there holds, for each agent i and for each terminal history h of agent i,

$$\begin{aligned}
\sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h,j')) \prod_{k \in I \setminus \{i\}} x_k^*(j_k') &= \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h,j')) \prod_{k \in I \setminus \{i\}} x_k^*(j_k') \Big\{ \frac{\sum_{h' \in \mathcal{E}_i} x_i^*(h')}{|O_i|^{T-1}} \Big\} \\
&= \frac{1}{|O_i|^{T-1}} \Big\{ \sum_{j' \in \mathcal{E}_{-i}} \sum_{h' \in \mathcal{E}_i} \mathcal{R}(\alpha, (h,j')) \prod_{k \in I \setminus \{i\}} x_k^*(j_k') x_i^*(h') \Big\} \\
&= \frac{1}{|O_i|^{T-1}} \Big\{ \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, (h, j_{-i})) \prod_{k \in I \setminus \{i\}} x_k^*(j_k') x_i^*(j_i') \Big\} \\
&= \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, (h, j_{-i})) z^*(j)
\end{aligned} \tag{5.89}$$

Thus, using this fact, we add (C5) to linearize the nonlinear term.

Due to (C1)-(C5), **NLP2** transforms to the following 0-1 MILP:

$$\text{Maximize} \quad y_1(\varnothing) \tag{5.90}$$

Subject To:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{5.91}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{5.92}$$

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{5.93}$$

$$y_i(\iota(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, (h, j_{-i}))z(j) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.94}$$

$$\sum_{j' \in \mathcal{E}_{-i}} z(h, j') = \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.95}$$

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{5.96}$$

$$x_i(h) \leq 1 - b_i(h), \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.97}$$

$$w_i(h) \leq \mathcal{U}_i(h) b_i(h), \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.98}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{5.99}$$

$$x_i(h) \in \{0, 1\} \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.100}$$

$$w_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.101}$$

$$b_i(h) \in \{0, 1\}, \quad \forall h \in \mathcal{H}_i \tag{5.102}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad \forall i \in I, \forall \iota \in \mathcal{I}_i \tag{5.103}$$

$$z(j) \in [0, 1], \quad \forall j \in \mathcal{E} \tag{5.104}$$

We shall henceforth refer to the 0-1 MILP (5.90)-(5.104) as **MILP4**.

The variables of the program are the vectors x_i, w_i, b_i and y_i for each agent i and the vector z. We have the following result.

Theorem 5.3. *Given a solution (x^*, w^*, y^*, b^*, z^*) to* **MILP4**, $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ *is a pure T-period optimal joint policy.*

Proof: Due to the policy constraints and the domain constraints of each agent, each x_i^* is a pure policy of agent i. Due to the constraints (5.93)-(5.94), each y_i^* contains the values of the information sets of agent i given x_{-i}^*. Due to the complementarity constraints (5.97)-(5.98), each x_i^* is a best response to x_{-i}^*. Thus x^* is a Nash equilibrium. Finally, by maximizing the value of the null information set of agent 1, we are effectively maximizing the value of x^*. Thus x^* is an optimal joint policy. *Q.E.D*

Thus, till now we have presented three 0-1 MILPs which can be used to find an optimal T-period joint policy for 2, 3 or more agents, **MILP1**, **MILP2** and now **MILP4**. How do the three compare? As explained in Chapter 4, **MILP2** is better (smaller) than **MILP1**, and it also has exponential lesser 0-1 variables than it. Therefore, it is interesting to rather compare

MILP2 with **MILP4**. Comparing the two, we see that sizes of the two programs are more or less the same, but the latter does have more variables and more 0-1 variables than the former. To be precise, **MILP2** has a 0-1 variable for every terminal history of every agent while **MILP4** has two 0-1 variables for every terminal as well as nonterminal history of each agent.

5.7.1 An Alternative 0-1 MILP

So, from the preceding discussion it is clear that for finding an optimal joint policy when the number of agents is more than 2, the smallest 0-1 MILP is **MILP2**, presented in Chapter 4. The number of 0-1 variables in this exponential in T. However, the total number of variables in it is exponential in T and in n. This is also the case with **MILP4** presented above. In this section, we present a slightly different 0-1 MILP which has more 0-1 variables than**MILP4** but overall has lesser variables than **MILP4** and in fact even than **MILP2**.

The 0-1 MILP proposed in this section is obtained by making changes and additions to **NLP2** just as **MILP4** was conceived. The main difference between the proposed 0-1 MILP and **MILP4** is that while the latter has a continuous variable for every terminal joint history, the former will have a 0-1 variable for every reduced terminal joint history. Since the number of reduced terminal joint histories is lesser than the number of terminal joint histories, the former is of a smaller size.

The additions/changes to **NLP2** are as follows.

(D1) (C1) as in in Section (5.7).

(D2) For each agent $i \in I$, for each i-reduced terminal joint history $j' \in \mathcal{E}_{-i}$, we add a 0-1 variable $z_{-i}(j')$:

$$z_{-i}(j') \quad \in \quad \{0,1\}, \quad \forall j' \in \mathcal{E}_{-i} \tag{5.105}$$

(D3) For each agent $i \in I$, we replace the nonlinear equations (5.51) by the linear equations

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, (h, j'))z_{-i}(j') \quad = \quad w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.106}$$

(D4) For each agent $i \in I$, for each agent $k \neq i$, we add the constraints,

$$\sum_{j' \in \mathcal{E}_{-i,-k}} z_{-i}(h, j') \quad = \quad \prod_{l \in I \backslash \{i,k\}} |O_l|^{T-1} x_k(h), \quad \forall h \in \mathcal{E}_k \tag{5.107}$$

where $\mathcal{E}_{-i,-k}$ denotes the set of (i, k)-reduced terminal joint histories; an (i, k)-reduced terminal joint history is an (n - 2)-tuple of terminal histories in which the histories of agents i and k are missing.

(D5) For each agent $i \in I$, we add the constraint,

$$\sum_{j' \in \mathcal{E}_{-i}} z_{-i}(j') \quad = \quad \prod_{k \in I \backslash \{i\}} |O_k|^{T-1} \tag{5.108}$$

The effects of these changes/additions are as follows. Due to (D1), every solution x^* to the resultant program is a pure joint policy; each x_i^* is a pure policy of agent i. (D2) allows us to linearize the NLP; (D3)-(D5) ensure that the following double implication holds for each agent $i \in I$ and for each i-reduced terminal joint history $j' \in \mathcal{E}_{-i}$,

$$z_{-i}^*(j') = 1 \quad \Leftrightarrow \quad x_k^*(j_k') = 1, \quad \forall k \in I \backslash \{i\} \tag{5.109}$$

With (D1)-(D5), **NLP2** changes to the following 0-1 MILP:

$$\text{Maximize} \quad y_1(\varnothing) \tag{5.110}$$

Subject To:

$$\sum_{a\in A_i} x_i(a) = 1 \tag{5.111}$$

$$-x_i(h) + \sum_{a\in A_i} x_i(hoa) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{5.112}$$

$$y_i(\iota(h)) - \sum_{o\in O_i} y_i(ho) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{5.113}$$

$$y_i(\iota(h)) - \sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,(h,j'))z_{-i}(j') = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.114}$$

$$\sum_{j'\in\mathcal{E}_{-i,-k}} z_{-i}(h,j') = \prod_{l\in I\backslash\{i,k\}} |O_l|^{T-1} x_k(h), \quad \forall i \in I, \forall k \in I\backslash\{i\}, \forall h \in \mathcal{E}_k \tag{5.115}$$

$$\sum_{j'\in\mathcal{E}_{-i}} z_{-i}(j') = \prod_{k\in I\backslash\{i\}} |O_k|^{T-1}, \quad \forall i \in I \tag{5.116}$$

$$x_i(h) \leq 1 - b_i(h), \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.117}$$

$$w_i(h) \leq \mathcal{U}_i(h) b_i(h), \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.118}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{5.119}$$

$$x_i(h) \in \{0,1\} \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{5.120}$$

$$w_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{5.121}$$

$$b_i(h) \in \{0,1\}, \quad \forall h \in \mathcal{H}_i \tag{5.122}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad \forall i \in I, \forall \iota \in \mathcal{I}_i \tag{5.123}$$

$$z_{-i}(j') \in \{0,1\}, \quad \forall i \in I, \forall j' \in \mathcal{E}_{-i} \tag{5.124}$$

We shall henceforth refer to the 0-1 MILP (5.110)-(5.124) as **MILP5**.

Lemma 5.1. *Every solution (x^*, y^*, z^*, w^*, b^*) to* **MILP5** *satisfies the following condition for every agent i and for every i-reduced terminal joint history $j' \in \mathcal{E}_{-i}$,*

$$z^*(j') = 1 \quad \Leftrightarrow \quad x_k^*(j'_k) = 1, \quad \forall k \in I\backslash\{i\} \tag{5.125}$$

Proof: The proof is completely analogous to the proof of Lemma (4.2), Chapter 4. We give it for the sake of completeness. Let (x^*, y^*, z^*, w^*, b^*) denote a solution to **MILP5**. Let,

$$S_{-i}(z) = \{j' \in \mathcal{E}_{-i} | z^*(j') = 1\}, \quad \forall i \in I \tag{5.126}$$

$$S(x_i) = \{h \in \mathcal{E}_i | x_i^*(h) = 1\}, \quad \forall i \in I \tag{5.127}$$

$$S_{-i}(x) = \{j' \in \mathcal{E}_{-i} | x_k^*(j'_k) = 1, \forall k \in I\backslash\{i\}\} \tag{5.128}$$

If we show that $S_{-i}(x)$ is identical to $S_{-i}(z)$, then the statement of the lemma is proved.

Since each z variable in the program and each x variable in the program corresponding to a terminal history is a 0-1 variable, due to (5.115), the following implication clearly holds for each agent $i \in I$ and for each i-reduced terminal joint history j',

$$z^*(j') = 1 \quad \Rightarrow \quad x^*_k(j'_k) = 1, \quad \forall k \in I \backslash \{i\} \tag{5.129}$$

Hence, there holds for each agent $i \in I$,

$$S_{-i}(z) \quad \subseteq \quad S_{-i}(x) \tag{5.130}$$

Now, for each agent $i \in I$, x^*_i is a pure policy due (3.4) of Chapter 3. Therefore, for each agent $i \in I$, $|S_{-i}(x)| = \prod_{k \in I \backslash \{i\}} |O_k|^{T-1}$. Since each z variable is also a 0-1 variable, due to (5.116), there holds $|S_{-i}(z)| = \prod_{k \in I \backslash \{i\}} |O_k|^{T-1}$. Thus, for each agent $i \in I$, there holds,

$$|S_{-i}(z)| \quad = \quad |S_{-i}(x)| \tag{5.131}$$

By (5.130) and (5.131), for each agent $i \in I$, there holds that $S_{-i}(z) = S_{-i}(x)$. This proves the statement of the lemma. *Q.E.D*

We thus have the following result.

Theorem 5.4. *Given a solution (x^*, w^*, y^*, b^*, z^*) to* **MILP5**, *$x^* = (x^*_1, x^*_2, \ldots, x^*_n)$ is a pure T-period optimal joint policy.*

Proof: The proof is analogous to the proof of Theorem (5.3). *Q.E.D*

5.8 Summary

In this chapter, we have presented three 0-1 MILPs, namely **MILP3**, **MILP4** and **MILP5**, for finding an optimal T-period joint policy. **MILP3** is conceived using the property that every optimal joint policy is also a Nash Equilibrium and it is limited to the 2-agent case. **MILP4** and **MILP5** are conceived using the properties that every optimal joint policy is also a Nash Equilibrium and an optimal joint policy can be pure, and they are conceived essentially for the 3 or more agents case, but they are also applicable to the 2-agent case. Of the five 0-1 MILPs presented in Chapters 4 and 5, **MILP3** is the smallest.

The 0-1 MILPs presented in this chapter enjoy an additional property which the 0-1 MILPs presented in Chapter 4 do not, and it is that they can be adapted for solving partially observable stochastic games (POSGs). **MILP3** can be easily modified (by removing its objective function) to find a sample Nash Equilibrium of a 2-agent DEC-POMDP. Alternatively, it can be adapted to find a sample Nash Equilibrium of a 2-player extensive game, thus providing an interesting alternative to the KMvS approach [KMvS94] of solving an LCP. Moreover, **MILP3** has an advantage over the KMvS approach in that it can be used to find a socially maximizing Nash Equilibrium of a 2-agent POSG/2-player extensive game, something which the KMvS approach is not capable of doing. Annex (A) describes how **MILP3** can be adapted to solve POSGs.

MILP4 and **MILP5** too can be used for solving POSGs but for a special case only. They can be used to find a sample Nash Equilibrium of a 3-or-more-agents POSG/3-or-more-players extensive game only if the POSG/extensive game has a pure Nash Equilibrium. POSGs or games which have pure Nash Equilibrium are not necessarily rare, but are not guaranteed to exist for each POSG/game either. On the other hand, even for 3+ agents/players POSG/games which do have a pure Nash Equilibrium, there isn't a wide variety of algorithms capable of finding it. In fact, the only algorithm we are aware of is the Govindan Wilson Algorithm [GW01]. Thus, **MILP4** and **MILP5** may have some use in this direction as well.

In the next chapter, we present heuristics to reduce the space and time requirement of the five programs **MILP1** to **MILP5** presented in Chapter 4 and in this chapter.

Chapter 6

Heuristics And Dynamic Programming

6.1 Introduction

In this chapter, we present different ways in which the practical performance of the 0-1 MILPs **MILP1** to **MILP5** presented in Chapters 4 and 5 can be improved.

The size of each of the programs is a function of the sizes of the n sets of histories $\mathcal{H}_1$, $\mathcal{H}_2$, ..., $\mathcal{H}_n$. The number of variables and constraints of a program is in direct proportion to the sizes of these sets. The size of each of these sets of exponential in the horizon T. The size of a program can be reduced if we remove all *extraneous* histories from it. We define two types of extraneous histories, *locally extraneous* histories and *globally extraneous* histories.

A locally extraneous history is defined for the initial state α. If a history is locally extraneous at α then it means that it is provably not required in order to find an optimal joint policy at α. A globally extraneous history, on the other hand, is provably not required in order to find an optimal joint policy *whatever* be the initial state of the DEC-POMDP (including evidently α). If a history is extraneous, in formulating a program we do not require a variable for it. We present two dynamic programming algorithms to prune extraneous histories, one that prunes locally extraneous histories and the other that prunes globally extraneous ones. The method that prunes globally extraneous is similar to the backward induction DP Algorithm [HBZ04] that prunes extraneous (i.e., very weakly dominated) policies in the canonical form. Note that it is also possible to prune locally extraneous histories using a forward search algorithm such as A*.

To reduce the time requirement of solving a 0-1 MILP, we introduce constraints that bound the value of the objective either from below or above. For the upper bound, we use the value of the optimal policy of the corresponding POMDP. For the lower bound, we use the value of the optimal joint policy for horizon T - 1 in solving for an optimal joint policy for horizon T. Such constraints are called *cuts* because they carve out portions of the space of feasible solutions of a program which provably do not contain an optimal solution. They essentially guide the branch and bound (BB) method in achieving an early termination.

6.2 Locally Extraneous Histories

A locally extraneous history is a history that is not required to find an optimal joint policy at α because if it is in a policy that is a part of an optimal joint policy at α, it can be replaced by a *co-history* without affecting the value of the joint policy. A **co-history** of a history h of an agent is defined to be a history of that agent that is identical to h in all aspects except for its last action. The last action of the co-history is different from that of the history. If $A_i = \{b, c\}$, the co-history of *cubvb* is the history *cubvc*. The set of cohistories of a history h shall be denoted by $C(h)$.

A history $h \in \mathcal{H}_i^t$ of length t of agent i is said to be **locally extraneous** if for every probability distribution γ over the set $\mathcal{H}_{-i}^t$ of i-reduced joint histories of length t, there exists a history $h' \in C(h)$ such that,

$$\sum_{j' \in \mathcal{H}_{-i}^t} \gamma(j') \big\{ \mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j')) \big\} \quad \geq \quad 0 \tag{6.1}$$

where $\gamma(j')$ denotes the probability of j' in γ.

An alternative definition is as follows. A history $h \in \mathcal{H}_i^t$ of length t of agent i is said to be **locally extraneous** if there exists a probability distribution ω over the set of co-histories of h such that for each i-reduced joint history j' of length t, there holds,

$$\sum_{h' \in C(h)} \omega(h') \mathcal{R}(\alpha, (h', j')) \geq \mathcal{R}(\alpha, (h, j')) \tag{6.2}$$

where $\omega(h')$ denotes the probability of the co-history h' in ω.

The following theorem shows that in an optimal T-period joint policy, a locally extraneous terminal history receiving a nonzero weight can be replaced by *some* co-history with the same weight, without affecting the optimality of the joint policy.

Theorem 6.1. *For every optimal T-period joint policy p' such that for some agent $i \in I$ and for a terminal history h of agent i that is locally extraneous at α, $p'_i(h) > 0$, there exists another T-period joint policy p that is optimal at α and that is identical to p' in all respects except that $p_i(h) = 0$.*

Proof: Let p' be a T-period joint policy that is optimal at α. Assume that for some agent $i \in I$ and for a terminal history h of agent i that is locally extraneous at α, $p'_i(h) > 0$. By (6.1), there exists at least one co-history h' of h such that,

$$\sum_{j' \in \mathcal{H}_{-i}^T} p'_{-i}(j') \big\{ \mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j')) \big\} \quad \geq \quad 0 \tag{6.3}$$

Let q be a T-period policy of agent i that is identical to p'_i in all respects except that $q(h') =$

$p'_i(h) + p'_i(h')$ and $q(h) = 0$. We shall show that q is also optimal at α. There holds,

$$\begin{aligned}
\mathcal{V}(\alpha,(q,p'_{-i})) - \mathcal{V}(\alpha,(p'_i,p_{-i})) &= \\
\sum_{j'\in\mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha,(h',j'))q(h') - \mathcal{R}(\alpha,(h',j'))p'_i(h') - \mathcal{R}(\alpha,(h,j'))p'_i(h)\big\} &= \\
\sum_{j'\in\mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha,(h',j'))(q(h') - p'_i(h')) - \mathcal{R}(\alpha,(h,j'))p'_i(h)\big\} &= \\
\sum_{j'\in\mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha,(h',j'))p'_i(h) - \mathcal{R}(\alpha,(h,j'))p'_i(h)\big\}
\end{aligned}$$

since $q(h') = p'_i(h) + p'_i(h')$. Therefore,

$$\begin{aligned}
\mathcal{V}(\alpha,(q,p'_{-i})) - \mathcal{V}(\alpha,(p'_i,p_{-i})) &= \\
\sum_{j'\in\mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha,(h',j')) - \mathcal{R}(\alpha,(h,j'))\big\} &\geq 0 \quad \text{(due to (6.3))}
\end{aligned}$$

Hence, $p = (q,\, p'_{-i})$ is also an optimal T-period joint policy at α. *Q.E.D.*

The implication of Theorem (6.1) is understood as follows. If a history h is locally extraneous at α, one of the following two cases arises:

Case (i) Every co-history of h is also locally extraneous at α.

Case (ii) There is at least one co-history of h that is not locally extraneous at α.

For a history h of agent i let $\mathcal{C}(h)$ denote the set $\cup_{h'\in C(h)}C(h')$. $\mathcal{C}(h)$ is thus the set of all histories of agent i that are pairwise co-histories of one another . If a terminal history h of agent i is locally extraneous at α, then in order to find an optimal T-period joint policy at α using any of the mathematical programs presented in Chapters 4 and 5, Theorem (6.1) has the following implication:

- If Case (i) holds, at most one history from the set $\mathcal{C}(h)$ is required; other histories are not required, and thereby they do not need to be represented in a program through variables. Note that in this case, the word 'extraneous' is perhaps inappropriate since the history that we do retain from $\mathcal{C}(h)$ is also locally extraneous; in other words, we are retaining a history that is (if its co-histories were also retained) extraneous.
- If Case (ii) holds, h is not required; it need not be represented in a program through a variable.

6.3 Identifying Locally Extraneous Histories

We shall now define tests that identify if a given history is locally extraneous or not. There are two tests that we can perform. The first test is simple and is as follows. If the *a priori* probability of every joint history of length t of which the given history is a part of, occurring from α is 0, then the given history is locally extraneous. That is, a history h of length t of agent i is locally extraneous if there holds,

$$\Psi(\alpha,(h,j')) = 0, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{6.4}$$

Thus, to check is a history h is locally extraneous or not, we must verify if (6.4) is true for it or not. Note that if (6.4) is true for h then it is true for every co-history of h as well because a history differs from a co-history in only its last action, and in the definition of $\Psi(\alpha, (h, j'))$ (Chapter 3), the last joint action of the joint history $(h,\ j')$ is not taken into account. Now,

$$\mathcal{R}(\alpha, (h, j')) \quad = \quad \Psi(\alpha, (h, j'))\mathcal{S}(\alpha, (h, j')) \tag{6.5}$$

Therefore, if (6.4) is true for h, then for every $j' \in \mathcal{H}^t_{-i}$, $\mathcal{R}(\alpha, (h, j')) = 0$ and for every co-history h' of h, $\mathcal{R}(\alpha, (h', j')) = 0$. Hence, h is locally extraneous. Moreover, if (6.4) is true for h, then it is true for every co-history of h, and thereby every co-history of h is is also locally extraneous.

This test is not adequate. Even if (6.4) does not hold for h, h may still be locally extraneous. However, in many instances, this test is very fruitful; a large percentage of histories are identified as being locally extraneous just on the basis of this test.

The second test involves solving a linear program. The linear program is based on the definition of a locally extraneous history. So we have two choices. We may either solve an LP based on the first definition (that is, in which h is required to satisfy (6.1)) or we may solve an LP based on the second definition (that is, in which the history is required to satisfy (6.2)). The two LPs in fact form a primal-dual LP pair. The first LP is as follows. It determines if a history h of length t of agent i is locally extraneous or not.

$$\text{Minimize} \quad \epsilon \tag{6.6}$$

Subject To,

$$\sum_{j' \in \mathcal{H}^t_{-i}} y(j')\big\{\mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j'))\big\} \quad \leq \quad \epsilon, \quad \forall h' \in C(h) \tag{6.7}$$

$$\sum_{j' \in \mathcal{H}^t_{-i}} y(j') \quad = \quad 1 \tag{6.8}$$

$$y(j') \quad \geq \quad 0, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{6.9}$$

The LP contains one variable $y(j')$ for each i-reduced joint history j' of length t, constrained to be nonnegative. It has one variable, ϵ, which is free. Each variable $y(j')$ represents the probability of the i-reduced joint history j'. As the following lemma proves, if $\epsilon^* \geq 0$, then h is locally extraneous. We shall henceforth refer to the LP (6.6)-(6.9) as **LP1**.

Lemma 6.1. *If in a solution to (ϵ^*, y^*) to* **LP1**, $\epsilon^* \geq 0$, *then h is locally extraneous.*

Proof: Let (ϵ^*, y^*) be a solution to **LP1**. The value ϵ^* is in fact a solution to the following problem,

$$\epsilon^* \quad = \quad \min_{y \in \Delta(\mathcal{H}^t_{-i})} \max_{h' \in C(h)} \sum_{j' \in \mathcal{H}^t_{-i}} y(j')\big\{\mathcal{R}(\alpha, (h', j')) - \mathcal{R}(\alpha, (h, j'))\big\} \tag{6.10}$$

In other words, the LP finds a weight distribution $y^* \in \Delta(\mathcal{H}^t_{-i})$ where the difference ϵ^* between the value of any cohistory h' of h and the value of h is minimized. If this difference is negative, then clearly h is not locally extraneous at α when the other agents' policies have the weight distribution y^*. If on the other hand, this difference is 0 or positive, by definition, h is locally

extraneous at α if the other agents' policies have the weight distribution y^*. Moreover, this difference is *minimal* when the other agents' policies have the weight distribution y^*; for other weight distributions, it is as large as or is larger than ϵ^*, implying that h is certainly extraneous at α for other weight distributions as well. Thus, if $\epsilon^* \geq 0$, then h is locally extraneous at α. *Q.E.D*

An LP for checking if a history is locally extraneous or not that is based on the second definition is as follows.

$$\text{Maximize} \quad \epsilon \tag{6.11}$$

Subject To,

$$\sum_{h' \in C(h)} y(h')\mathcal{R}(\alpha,(h',j')) - \mathcal{R}(\alpha,(h,j')) \geq \epsilon, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{6.12}$$

$$\sum_{h' \in C(h)} y(h') = 1 \tag{6.13}$$

$$y(h') \geq 0, \quad \forall h' \in C(h) \tag{6.14}$$

If $\epsilon^* \geq 0$, then history h of length t of agent i is an extraneous history. The proof that this is so is analogous to the proof of Lemma (6.1). We shall henceforth refer to this LP as **LP2**.

6.3.1 Pruning Locally Extraneous Terminal Histories

The following procedure identifies all locally extraneous terminal histories of all the agents. The procedure is similar to the procedure of iterated elimination of dominated strategies in a game [OR94].

Step 1: For each agent $i \in I$, set $\tilde{H}^T_i$ to $\mathcal{E}_i$. Let $\tilde{H}^T$ denote the set $\times_{i \in I} \tilde{H}^T_i$. For each joint history $j \in H^T$, compute and store the value $\mathcal{R}(\alpha, j)$ of j and the joint observation sequence probability $\Psi(\alpha, j)$ of j.

Step 2: For each agent $i \in I$, for each history $h \in \tilde{H}^T_i$, if for each i-reduced joint history $j' \in \tilde{H}^T_{-i}$, $\Psi(\alpha,(h,j')) = 0$, remove h from $\tilde{H}^T_i$.

Step 3: For each agent $i \in I$, for each history $h \in \tilde{H}^T_i$ do as follows: If $C(h) \cap \tilde{H}^T_i$ is non-empty, check if h is locally extraneous or not by setting up and solving **LP1** or **LP2**. When setting up either LP, replace $\mathcal{H}^t_{-i}$ by the set $\tilde{H}^T_{-i}$ and the set $C(h)$ by the set $C(h) \cap \tilde{H}^T_i$. If upon solving the LP, h is found to be locally extraneous at α, remove h from $\tilde{H}^T_i$.

Step 4: If in Step 3 a history (of any agent) is found to be locally extraneous, go to Step 3. Otherwise, terminate the procedure.

The procedure produces the set $\tilde{H}^T_i$ for each agent i. This set contains every terminal history of agent i that is required for finding an optimal joint policy at α, that is every terminal history that is not locally extraneous at α. For each agent i, every history that is in $\mathcal{H}^T_i$ but not in $\tilde{H}^T_i$ is locally extraneous. The reason for reiterating Step 3 is that if a history h of some agent i is

found to be locally extraneous and consequently removed from $\tilde{H}_i^T$, it is possible that a history of some other agent that was previously not locally extraneous now becomes so, due to the removal of h from $\tilde{H}_i^T$. Hence, in order to verify if this is the case for any history or not, we reiterate Step 3.

6.3.2 Pruning All Locally Extraneous Histories

As proved in Theorem (6.1), we can always find a T-period optimal joint policy at α without requiring locally extraneous terminal histories. This also implies that we do not require all those non-terminal histories that give rise only to locally extraneous terminal histories. For instance, if a history h of length T - 1 of agent i is such that for each observation $o \in O_i$ and for each action $a \in A_i$, the terminal history hoa is locally extraneous at α, then in effect, h is not required to find an optimal joint policy at α. In other words h itself can be considered as locally extraneous. A locally extraneous history can be defined recursively as follows. A history h of length t of agent i is **locally extraneous** at α if for each observation $o \in O_i$ and for each action $a \in A_i$, the history hoa is locally extraneous at α. The following procedure identifies all locally extraneous of all lengths from 1 to T of all the agents.

Step 1: For each agent $i \in I$, construct the set $\tilde{H}_i^T$ containing all terminal histories that are not locally extraneous at α using the Pruning Locally Extraneous Terminal Histories procedure given in Section (6.3.1). Set t to T - 1.

Step 2: For each agent $i \in I$, do as follows: Set $\tilde{H}_i^t$ to $\mathcal{H}_i^t$. For each history $h \in \tilde{H}_i^t$, if $\tilde{H}_i^{t+1}$ does not contain even one **child** of h, then remove h from $\tilde{H}_i^t$. The set of children of a history h of agent i is defined as $\{hoa | a \in A_i, o \in O_i\}$.

Step 3: Decrement t by 1. If $t > 0$, go to Step 2. Otherwise, terminate the procedure.

For each agent $i \in I$, the procedure creates T sets, $\tilde{H}_i^T \subseteq \mathcal{H}_i^T$, $\tilde{H}_i^{T-1} \subseteq \mathcal{H}_i^{T-1}$, ..., $\tilde{H}_i^1 \subseteq \mathcal{H}_i^1$ in that order. Each set $\tilde{H}_i^t$ contains all histories of length t of agent i that are not locally extraneous at α. Note that a set $\tilde{H}_i^t$ may contain a history that is locally extraneous at α if and only if every co-history of that history is also locally extraneous at α.

6.4 Globally Extraneous Histories

A globally extraneous history is not required to find an optimal joint policy for *any* initial state of the DEC-POMDP including the given one, α. That is, for any initial state $\beta \in \Delta(S)$, we can find an optimal joint policy at β in which only histories that are not globally extraneous receive a nonzero weight. Thus, unlike a locally extraneous history, a globally extraneous history is not a function of a particular state in $\Delta(S)$. When a history is globally extraneous, it is extraneous for *all* the states in the simplex $\Delta(S)$. When a history is not globally extraneous, it means that there is at least one $\beta \in \Delta(S)$ such that in an optimal joint policy at β, the history *definitely* receives a nonzero weight.

The definition of a globally extraneous history is similar to that of a locally extraneous history. It is based on the fact that (as proved in Chapter 3, Section (3.5.1) the value of a joint history at a state in the interior of the simplex $\Delta(S)$ can be expressed as a convex combination of the

values of the joint history at the corners of $\Delta(S)$ (i.e., members of S). That is, for each joint history $j \in \mathcal{H}$ and for each $\beta \in \Delta(S)$, there holds,

$$\mathcal{R}(\beta, j) \quad = \quad \sum_{s \in S} \beta(s) \mathcal{R}(s, j) \tag{6.15}$$

A history $h \in \mathcal{H}_i^t$ of length t of agent i is said to be **globally extraneous** if for every probability distribution γ over the Cartesian set $\mathcal{H}_{-i}^t \times S$ there exists a co-history h' of h such that,

$$\sum_{s \in S} \sum_{j' \in \mathcal{H}_{-i}^t} \gamma(j', s) \big\{ \mathcal{R}(s, (h', j')) - \mathcal{R}(s, (h, j')) \big\} \quad \geq \quad 0 \tag{6.16}$$

where $\gamma(j', s)$ denotes the probability of the pair $(j',\ s)$ in γ.

An alternative definition is as follows. A history $h \in \mathcal{H}_i^t$ of length t of agent i is said to be **globally extraneous** if there exists a probability distribution ω over the set of co-histories of h such that for each i-reduced joint history j' of length t and for every state $s \in S$, there holds,

$$\sum_{h' \in C(h)} \omega(h') \mathcal{R}(s, (h', j')) \geq \mathcal{R}(s, (h, j')) \tag{6.17}$$

where $\omega(h')$ denotes the probability of the co-history h' in ω.

We therefore have the following result, similar to Theorem (6.1).

Theorem 6.2. *For any state $\beta \in \Delta(S)$, for every t-period joint policy p' optimal at β such that for some agent $i \in I$ and for a history h of length t of agent i that is globally extraneous, $p'_i(h) > 0$, there exists another t-period joint policy p that is optimal at α and that is identical to p' in all respects except that $p_i(h) = 0$.*

Proof: The proof is similar to the proof of Theorem (6.1). Let p' be an optimal t-period joint policy at β. Assume that for some agent i and for history h of length t of agent i, $p'_i(h) > 0$. Assume that h is globally extraneous. Then, for some co-history h' of h there holds,

$$\sum_{s \in S} \sum_{j' \in \mathcal{H}_{-i}^t} \beta(s) p'_{-i}(j') \big\{ \mathcal{R}(s, (h', j')) - \mathcal{R}(s, (h, j')) \big\} \quad \geq \quad 0 \tag{6.18}$$

Let q be a t-period policy of agent i that is identical to p'_i in all respects except that $q(h') = p'_i(h) + p'_i(h')$ and $q(h) = 0$. Then,

$$\begin{aligned}
\mathcal{V}(\beta, (q, p'_{-i})) - \mathcal{V}(\beta, (p'_i, p'_{-i})) \quad &= \\
\sum_{j' \in \mathcal{H}_{-i}^t} p'_{-i}(j') \big\{ \mathcal{R}(\beta, (h', j')) q(h') - \mathcal{R}(\beta, (h', j')) p'_i(h') - \mathcal{R}(\beta, (h, j')) p'_i(h) \big\} \quad &= \\
\sum_{j' \in \mathcal{H}_{-i}^t} p'_{-i}(j') \big\{ \mathcal{R}(\beta, (h', j')) (q(h') - p'_i(h')) - \mathcal{R}(\beta, (h, j')) p'_i(h) \big\} \quad &= \\
\sum_{j' \in \mathcal{H}_{-i}^t} p'_{-i}(j') \big\{ \mathcal{R}(\beta, (h', j')) p'_i(h) - \mathcal{R}(\beta, (h, j')) p'_i(h) \big\} \quad &= \\
\sum_{j' \in \mathcal{H}_{-i}^t} p'_{-i}(j') \big\{ \sum_{s \in S} \beta(s) \mathcal{R}(s, (h', j')) p'_i(h) - \sum_{s \in S} \beta(s) \mathcal{R}(s, (h, j')) p'_i(h) \big\}
\end{aligned}$$

since $q(h') = p'_i(h) + p'_i(h')$. Therefore,

$$\begin{aligned}
\mathcal{V}(\beta,(q,p'_{-i})) - \mathcal{V}(\beta,(p'_i,p'_{-i})) &= \\
\sum_{j'\in\mathcal{H}^t_{-i}} p'_{-i}(j') \sum_{s\in S} \beta(s)\big\{\mathcal{R}(s,(h',j'))p'_i(h) - \mathcal{R}(s,(h,j'))p'_i(h)\big\} &= \\
\sum_{s\in S}\sum_{j'\in\mathcal{H}^t_{-i}} \beta(s)p'_{-i}(j')\big\{\mathcal{R}(s,(h',j'))p'_i(h) - \mathcal{R}(s,(h,j'))p'_i(h)\big\} &\geq 0 \quad \text{(due to (6.18))}
\end{aligned}$$

Hence, $p = (q,\, p'_{-i})$ is also optimal at β. *Q.E.D.*

The implication of Theorem (6.2) is analogous to that of Theorem (6.1). If a history h is globally extraneous, one of the following two cases arises:

Case (i) Every co-history of h is also globally extraneous.

Case (ii) There is at least one co-history of h that is not globally extraneous.

If a history h of length t agent i is globally extraneous, then in order to find an optimal t-period joint policy at for any initial state $\beta \in \Delta(S)$, using any of the mathematical programs presented in Chapters 4 and 5, Theorem (6.2) has the following implication:

- If Case (i) holds, at most one history from the set $\mathcal{C}(h)$ is required; other histories are not required, and thereby they do not need to be represented in a program through variables.
- If Case (ii) holds, h is not required; it need not be represented in a program through a variable.

6.5 Identifying Globally Extraneous Histories

The tests for determining if a history is globally extraneous or not are analogous to those for determining if it is locally extraneous at α. We may check its joint observation sequence probabilities (with reduced joint histories) and solve a linear program. Thus, a history h of length t of agent i is globally extraneous if there holds,

$$\Psi(s,(h,j')) = 0, \quad \forall s \in S,\, \forall j' \in \mathcal{H}^t_{-i} \tag{6.19}$$

The converse of this is not necessarily true. Even if (6.19) does not hold, h may be globally extraneous. The following linear program determines if a history h of length t of agent i is globally extraneous or not. The LP is analogous to **LP1**.

$$\text{Minimize} \quad \epsilon \tag{6.20}$$

Subject To,

$$\sum_{s\in S}\sum_{j'\in\mathcal{H}^t_{-i}} y(s,j')\big\{\mathcal{R}(s,(h',j')) - \mathcal{R}(s,(h,j'))\big\} \leq \epsilon, \quad \forall h' \in C(h) \tag{6.21}$$

$$\sum_{j'\in\mathcal{H}^t_{-i}}\sum_{s\in S} y(j',s) = 1 \tag{6.22}$$

$$y(j',s) \geq 0, \quad \forall j' \in \mathcal{H}^t_{-i},\, \forall s \in S \tag{6.23}$$

If $\epsilon^* \geq 0$, then h is a globally extraneous history. We shall henceforth refer to LP (6.20)-(6.23) as **LP3**.

Lemma 6.2. *If in a solution to* (ϵ^*, y^*) *to* **LP3**, $\epsilon^* \geq 0$, *then* h *is globally extraneous.*

Proof: The proof is entirely analogous to the proof of Lemma (6.1).

We can determine if a history is globally extraneous or not by using the dual of **LP3** as well.

6.5.1 Pruning All Globally Extraneous Histories

The following procedure employs backward induction dynamic programming to identify all globally extraneous histories of all lengths from 1 to T of all the agents. The procedure is analogous to the DP Algorithm [HBZ04]. While the DP Algorithm operates in the space of policies, this procedure operates in the space of histories. This renders the procedure both less space consuming and faster. It is less space consuming than the DP Algorithm because unlike the latter, at each step it backs up only histories of all the agents, and not policies (in the canonical form) of all the agents. This means that at each step, the space requirement of the procedure grows linearly and not exponentially as in the DP Algorithm. Due to this linear growth, the number of linear programs that must be run to determine globally extraneous histories also grows only linearly and not exponentially as in DP. This makes the procedure much faster as well.

Step 1: For each agent $i \in I$, set $\overline{H}_i^1$ to A_i. Set t to 1.

Step 2: For each joint history $j \in \times_{i \in I} \overline{H}_i^T$, for each state $s \in S$, compute and store the value $\mathcal{R}(s, j)$ of j and the joint observations sequence probability $\Psi(s, j)$ of j in some data structure.

Step 3: For each agent $i \in I$, for each history $h \in \overline{H}_i^t$, do as follows: If for each i-reduced joint history $j' \in \overline{H}_{-i}^t$ and for each state $s \in S$, $\Psi(s, (h, j')) = 0$, remove h from $\overline{H}_i^t$.

Step 4: For each agent $i \in I$, for each history $h \in \overline{H}_i^t$, do as follows: If $C(h) \cap \overline{H}_i^T$ is non-empty, check if h is globally extraneous or not by setting up and solving **LP3**. When setting up the LP, replace $\mathcal{H}_{-i}^t$ by the set $\overline{H}_{-i}^t$ and the set $C(h)$ by the set $C(h) \cap \overline{H}_i^t$. If upon solving the LP, h is found to be globally extraneous, remove h from $\overline{H}_i^t$.

Step 5: If in Step 4 a history (of any agent) is found to be globally extraneous, go to Step 4. Otherwise, go to Step 6.

Step 6: Increment t by 1. If $t > T$, terminate the procedure, otherwise do as follows. For each agent $i \in I$, set $\overline{H}_i^t$ to the empty set. Then, for each agent $i \in I$ and for each history $h \in \overline{H}_i^{t-1}$, add the set $\{aoh | a \in A_i, o \in O_i\}$ to $\overline{H}_i^t$. Go to Step 2.

For each agent $i \in I$, the procedure creates T sets, $\overline{H}_i^1 \subseteq \mathcal{H}_i^1$, $\overline{H}_i^2 \subseteq \mathcal{H}_i^2$, ..., $\overline{H}_i^T \subseteq \mathcal{H}_i^T$ in that order. Each set $\overline{H}_i^t$ contains all histories of length t of agent i that are not globally extraneous history. Note that a set $\overline{H}_i^t$ may contain a history that is globally extraneous if and only if every co-history of that history is also globally extraneous. Step 6 is analogous to the full backup operation of the DP Algorithm. The following theorem shows that backing up histories as given in Step 6 does not exclude any history of length of t that is not globally extraneous.

Theorem 6.3. *If a history h of agent i is globally extraneous, for each $a \in A_i$ and for each $o \in O_i$, the history aoh is also globally extraneous.*

Proof: Let h be a history of length t of agent i. If h is globally extraneous, then for every state in $\Delta(S)$, for every i-reduced joint history j' of length t, there exists at least one co-history h' of h such that the value (h', j') is at least as large as the value of (h, j'). Therefore, for every history aoh, for every state in $\Delta(S)$, for every i-reduced joint history j'' of length $t+1$, there exists at least one history aoh' where h' is a co-history of h that is not globally extraneous, such that the value of (aoh', j'') is at least as large as the value of (aoh, j''), since the expected reward due to the action a is common to aoh and aoh'. Hence, every history aoh is globally extraneous if h is globally extraneous. *Q.E.D.*

6.6 Changes To The Programs

We shall now see what changes are required to the 0-1 MILPs **MILP1** to **MILP5** if instead of the complete sets of histories, sets of histories from which locally extraneous histories or globally extraneous histories have been removed are used.

Let $\tilde{H}_i$ denote the set $\tilde{H}_i^1 \cup \tilde{H}_i^2 \cup \ldots \cup \tilde{H}_i^T$. $\tilde{H}_i$ is the set that contains all histories of lengths less than or equal to T of agent i that are not locally extraneous at α. If in any mathematical program, for each agent $i \in I$, instead of the complete set of histories $\mathcal{H}_i$, the set $\tilde{H}_i$ is used, the following changes must be made. Every occurrence of $\mathcal{H}_i$ and $\mathcal{E}_i$ in the program must be replaced by respectively by $\tilde{H}_i$ and $\tilde{H}_i^T$. Similarly, every occurrence of $\mathcal{E}$ must be replaced by $\tilde{H}^T$ where $\tilde{H}^T$ denotes $\times_{i \in I} \tilde{H}_i^T$.

Similarly, $\overline{H}_i$ denote the set $\overline{H}_i^1 \cup \overline{H}_i^2 \cup \ldots \cup \overline{H}_i^T$. $\overline{H}_i$ is the set that contains all histories of lengths less than or equal to T of agent i that are not globally extraneous. If in any mathematical program, for each agent $i \in I$, instead of $\mathcal{H}_i$, the set $\overline{H}_i$ is used, the following changes must be made. Every occurrence of $\mathcal{H}_i$ and $\mathcal{E}_i$ in the program must be replaced by respectively by $\overline{H}_i$ and $\overline{H}_i^T$. Similarly, every occurrence of $\mathcal{E}$ must be replaced by $\overline{H}^T$ where $\overline{H}^T$ denotes $\times_{i \in I} \overline{H}_i^T$.

The 0-1 MILPs **MILP1** and **MILP2** presented in Chapter 4 and **MILP4** and **MILP5** presented in Chapter 5, exploit the structure of a finite horizon policy, and they are subject to additional changes. In particular, these programs rely on the fact that the number of histories of a given length in the support of a pure policy of each agent is fixed. The number of histories of length t, $t \leq T$, in the support of a pure T-period policy of agent i is $|O_i|^{t-1}$. However, when locally or globally extraneous histories are removed, this may no longer hold true. That is, it is possible that the number of histories of length t in the support of every pure T-period policy is less than $|O_i|^{t-1}$. Accordingly, certain constraints in these programs must be modified.

Let H^T denote either $\tilde{H}^T$ or $\overline{H}^T$. Similarly, for each agent $i \in I$, let H_i^T denote either $\tilde{H}_i^T$ or $\overline{H}_i^T$. The following changes are to be made *only if* $|H^T| < |\mathcal{E}|$.

- In **MILP1**, **MILP2** and **MILP4**, the constraint,

$$\sum_{j \in \mathcal{E}} z(j) \;=\; \prod_{i \in I} |O_i|^{T-1}$$

must be replaced by,

$$\sum_{j \in H^T} z(j) \quad \leq \quad \prod_{i \in I} |O_i|^{T-1} \tag{6.24}$$

This is the only change to be made to the program **MILP1**.

- In **MILP2** and **MILP4**, an additional change is required. The set of constraints,

$$\sum_{j' \in \mathcal{E}_{-i}} z(h, j') \quad = \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i$$

must be replaced by,

$$\sum_{j' \in H^T_{-i}} z(h, j') \quad \leq \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in H^T_i \tag{6.25}$$

In **MILP4** one more change is required for each agent $i \in I$ for whom $|H^T_i| < |\mathcal{E}_i|$. For each agent $i \in I$, let $\tilde{O}^{T-1}_i \subseteq \overline{O}^{T-1}_i$ denote the set of sequences of T - 1 observations of agent i such that for each sequence $\overline{o} \in \tilde{O}^{T-1}_i$, there exists at least one history h of length T in H^T_i such that the sequence of observations of h is $\overline{o}$. Then, in **MILP5**, for each agent $i \in I$ for whom $|H^T_i| < |\mathcal{E}_i|$, the set of constraints,

$$y_i(\iota(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, (h, j_{-i})) z(j) \quad = \quad w_i(h), \quad \forall h \in \mathcal{E}_i$$

must be replaced by,

$$y_i(\iota(h)) - \frac{1}{|\tilde{O}^{T-1}_i|} \sum_{j \in H^T} \mathcal{R}(\alpha, (h, j_{-i})) z(j) \quad = \quad w_i(h), \quad \forall h \in H^T_i \tag{6.26}$$

In **MILP5**, the following changes are to be made for each agent $i \in I$ for whom $|H^T_{-i}| < |\mathcal{E}_{-i}|$.

- The constraint,

$$\sum_{j' \in \mathcal{E}_{-i}} z_{-i}(j') \quad = \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1}$$

must be replaced by,

$$\sum_{j' \in H^T_{-i}} z_{-i}(j') \quad \leq \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} \tag{6.27}$$

- The set of constraints,

$$\sum_{j' \in \mathcal{E}_{-i,-k}} z_{-i}(h, j') \quad = \quad \prod_{l \in I \setminus \{i,k\}} |O_l|^{T-1} x_k(h), \quad \forall k \in I \setminus \{i\}, \forall h \in H_k$$

must be replaced by,

$$\sum_{j' \in H^T_{-i,-k}} z_{-i}(h, j') \quad \leq \quad \prod_{l \in I \setminus \{i,k\}} |O_l|^{T-1} x_k(h), \quad \forall k \in I \setminus \{i\}, \forall h \in H_k \tag{6.28}$$

If we construct for each agent $i \in I$ the sets $\overline{H}_i$ of non globally extraneous histories using the procedure in Section (6.5.1), then this means that we have not computed the value $\mathcal{R}(\alpha, j)$ of each joint history $j \in \overline{H}_i$ at α. We need not compute this value since we know that for each $j \in \overline{H}_i$, $\mathcal{R}(\alpha, j)$ equals $\sum_{s \in S} \alpha(s)\mathcal{R}(s, j)$. Thus, if we construct a program using the sets $\overline{H}_i$ rather than the set $\mathcal{H}_i$, in **MILP1** and **MILP2**, the objective function can be either,

$$\text{Maximize} \qquad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j)$$

or it can simply be,

$$\text{Maximize} \qquad \sum_{j \in \overline{H}^T} \sum_{s \in S} \alpha(s)\mathcal{R}(s, j) z(j) \tag{6.29}$$

Similarly, in **MILP3**, **MILP4** and **MILP5**, for each terminal joint history $j \in \overline{H}^T$, the term $\mathcal{R}(\alpha, j)$ can be replaced by $\sum_{s \in S} \alpha(s)\mathcal{R}(s, j)$.

6.7 Adding Cuts

The heuristics presented in the previous sections were meant for reducing the size of the 0-1 MILPs presented in Chapters 4 and 5. Reducing the size of a program generally also decreases the time taken to solve the program. As described in Chapter 4, the time taken by the branch and bound (BB) method to solve a 0-1 MILP is a function of the number of 0-1 variables in it. By not requiring variables for locally or globally extraneous histories in a 0-1 MILP, we therefore also reduce the time taken by the BB method to solve the program.

In this section, we present an heuristic which is expressly meant to decrease the time taken to solve the 0-1 MILPs of Chapters 4 and 5. The heuristic we propose is the introduction of *cuts* into a program. A cut [Dan60] is a constraint that identifies a portion of the set of feasible solutions in which the optimal solution provably does not lie. We propose two cuts as follows.

6.7.1 Upper Bound On Value

The first cut we propose is the **upper bound POMDP cut**. The value of an optimal T-period joint policy at α is bounded from above by the value of an optimal T-period POMDP policy at α. Therefore, in **MILP1** and **MILP2**, the upper bound POMDP cut is the following constraint:

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \quad \leq \quad \mathcal{V}_P^*(\alpha) \tag{6.30}$$

where $\mathcal{V}_P^*(\alpha)$ denotes the value of the optimal T-period POMDP policy at α. For, **MILP3**, **MILP4** and **MILP5**, which have a different objective function, the upper bound POMDP cut is the following constraint:

$$y_1(\varnothing) \quad \leq \quad \mathcal{V}_P^*(\alpha) \tag{6.31}$$

Likewise, we can also add the upper bound MDP cut.

The motivation behind adding the upper bound POMDP cut is that computing $\mathcal{V}_P^*(\alpha)$ requires time that is minuscule compared to the time taken to solve any of the 0-1 MILPs given in Chapters 4 and 5; typically, it takes less than 1% of the time taken to any of the solve the 0-1 MILPs. The optimal T-period POMDP policy and thereby the value $\mathcal{V}_P^*(\alpha)$ is found by solving a linear program. A POMDP policy in the sequence form for the given DEC-POMDP is defined as follows.

Let $\mathcal{H}$ denote the set $\cup_{t=1}^{T}\mathcal{H}^t$ of joint histories of lengths less than or equal to T. Let $\mathcal{N}$ denote the set $\mathcal{H}\backslash\mathcal{E}$ of nonterminal joint histories. Then, we define a **T-period POMDP policy in the sequence-form** to be a function q from $\mathcal{H}$ to $[0, 1]$ such that,

$$\sum_{a\in A} q(a) = 1 \tag{6.32}$$

$$-q(j) + \sum_{a\in A} q(joa) = 0, \quad \forall j \in \mathcal{N}, \forall o \in O \tag{6.33}$$

where joa denotes the joint history obtained on concatenating the joint observation o and the joint action a to joint history j. Note that this definition is given in terms of joint actions and joint observations, but it applies analogously to a given POMDP.

The **value** $\mathcal{V}_P(\alpha, q)$ of a T-period POMDP policy q at α is defined as,

$$\mathcal{V}_P(\alpha, q) = \sum_{j\in\mathcal{E}} \mathcal{R}(\alpha, j)q(j) \tag{6.34}$$

Thereby, a linear program that finds an optimal T-period POMDP policy in the sequence form is as follows,

$$\text{Maximize} \quad \sum_{j\in\mathcal{E}} \mathcal{R}(\alpha, j)y(j) \tag{6.35}$$

Subject To,

$$\sum_{a\in A} y(a) = 1 \tag{6.36}$$

$$-y(j) + \sum_{a\in A} y(joa) = 0, \quad \forall j \in \mathcal{N}, \forall o \in O \tag{6.37}$$

$$y(j) \geq 0, \quad \forall j \in \mathcal{H} \tag{6.38}$$

This LP has a variable $y(j)$ for each joint history $j \in \mathcal{H}$. A solution y^* to the LP constitutes an optimal POMDP policy. This LP shall be henceforth be denoted by **LP4**.

Thereby,

$$\mathcal{V}_P^*(\alpha) = \sum_{j\in\mathcal{E}} \mathcal{R}(\alpha, j)y^*(j) \tag{6.39}$$

Note that if q^* is an optimal POMDP policy in the sequence form and $p^* = (p_1^*, p_2^*, \ldots, p_n^*)$ is an optimal DEC-POMDP joint policy in the sequence form, there holds,

$$\mathcal{V}(\alpha, p^*) \leq \mathcal{V}_P(\alpha, q^*) \tag{6.40}$$

6.7.2 Lower Bound On Value

If in a DEC-POMDP, the reward function consists of only nonnegative rewards, the value of an optimal T-period joint policy at α is bounded from below by the value of an optimal (T - 1)-period joint policy at α. In the case where the rewards are allowed to be negative, the value of an optimal T-period joint policy at α is bounded from below by the sum of the value of an optimal (T - 1)-period joint policy at α and the lowest 1-period reward obtainable by any joint action. Thus, the **lower bound DEC-POMDP cut** consists of adding the following constraint to **MILP1** and **MILP2**:

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \quad \geq \quad \mathcal{V}(\alpha, p^*) + \min_{a \in A} \min_{s \in S} R(s, a) \tag{6.41}$$

where p^* denotes the optimal (T - 1)-period joint policy at α, which itself may be obtained by solving any of the 0-1 MILPs (this sort of recursion is reminiscent of the Recursive MAA* Algorithm, i.e., of the MAA* using the DEC-POMDP heuristic, as described in Chapter 2).

Similarly, for **MILP3**, **MILP4** and **MILP5**, the lower bound DEC-POMDP cut is added as,

$$y_1(\varnothing) \quad \geq \quad \mathcal{V}(\alpha, p^*) + \min_{a \in A} \min_{s \in S} R(s, a) \tag{6.42}$$

$\mathcal{V}(\alpha, p^*)$ itself must be computed using any of the 0-1 MILPs **MILP1** to **MILP5**. Notice that the addition of this cut is analogous to the MAA* algorithm employing the DEC-POMDP heuristic.

6.7.3 Impact Of The Cuts

The addition of cuts in a program is meant principally to assists the branch and bound (BB) method in solving any of the 0-1 MILPs, **MILP1** to **MILP5**. As described in Section (4.4.2), Chapter 4, the BB method is required to solve every linear program (LP) present in the list of active subproblems. Note that solving an LP potentially adds more LPs to this list. However, due to the upper bound POMDP cut and the lower bound DEC-POMDP cut, the BB method may not required to add an LP to this list which it would otherwise be required to do for the following reason.

Recall that $\mathbf{L}_a$ denotes the active subproblem selected by the BB method in Step 3. Suppose that upon solving $\mathbf{L}_a$, case (*iii*) occurs. In this case, the BB method is required to select a branching variable x and add two LPs $\mathbf{L}_x^0$ and $\mathbf{L}_x^1$ to the list. Note that $\mathbf{L}_x^0$ and $\mathbf{L}_x^1$ are called the children of $\mathbf{L}_a$. For each $i = 0, 1$, let u_{L_i} denote the value of the objective function of $\mathbf{L}_x^i$ for an optimal solution to $\mathbf{L}_x^i$. Let u_{L_a} denote the value of the objective function of $\mathbf{L}_a$ for an optimal solution to $\mathbf{L}_a$. Then, the following property holds for every active subproblem $\mathbf{L}_a$ and its children:

$$u_{L_a} \quad \geq \quad u_{L_0} \tag{6.43}$$

$$u_{L_a} \quad \geq \quad u_{L_1} \tag{6.44}$$

Thus, every active subproblem has a value that is at least as high as any of its children. We are exploiting precisely this property in adding cuts.

The use of the upper bound POMDP cut is as follows. In Step 3, if case (*ii*) occurs, in the absence of cuts, we add the solution to $\mathbf{L}_a$ to the list of feasible solutions. Now suppose we add the upper bound POMDP cut and in Step 3, case (*ii*) occurs. If it happens that,

$$u_{L_a} = \mathcal{V}^*_P(\alpha) \tag{6.45}$$

then, the BB method can terminate because the solution to $\mathbf{L}_a$ is also a solution to $\mathbf{M}$. Thus, this cut helps an early termination to the BB method. In other words, it identifies another terminating condition for the BB method. Note that in most DEC-POMDPs, it is the case that for every active subproblem $\mathbf{L}_a$ that has a feasible solution,

$$u_{L_a} < \mathcal{V}^*_P(\alpha) \tag{6.46}$$

That is, the value of the POMDP policy is strictly larger than the value of the DEC-POMDP policy. In such cases, the upper bound DEC-POMDP cut has no impact on the run time of the BB method.

The use of the lower bound DEC-POMDP cut is as follows. In Step 3, if case (*iii*) occurs, in the absence of cuts, we choose a branching variable x and add the children $\mathbf{L}^0_x$ and $\mathbf{L}^1_x$ of $\mathbf{L}_a$ to the list of active subproblems. Now suppose we add the lower bound DEC-POMDP cut and in Step 3, case (*iii*) occurs. If it happens, for any $i = 0, 1$, that,

$$u_{L_i} < \mathcal{V}(\alpha, p^*) + \min_{a \in A} \min_{s \in S} R(s, a) \tag{6.47}$$

then, we clearly do not need to add $\mathbf{L}^i_x$ to the list of active subproblems.

Thus, the lower bound DEC-POMDP cut effectively *prunes* the tree of active subproblems in an anticipatory fashion. It prevent the tree from growing in directions where the optimal solution cannot possibly lie. It reduces the runtime of the BB method. Compared to the upper bound POMDP cut, the lower bound DEC-POMDP cut can be termed as more useful since in most DEC-POMDPs, the addition of this cut prunes the tree to some extent.

6.8 Summary

In this chapter we have proposed heuristics intended to reduce the space required to formulate the 0-1 MILPs presented in Chapters 4 and 5, as well as the time taken to solve them. The first set of heuristics consists of identifying extraneous histories. We have defined two types of extraneous histories, and given tests (linear programs) to ascertain if a history is extraneous or not. We have also defined dynamic programming procedures to identify all extraneous histories. Extraneous histories are not required in finding an optimal joint policy, and therefore we do not require variables to represent them in the 0-1 MILPs. The second set of heuristics consists of adding constraints to the programs that bound from above or below the objective function. We have proposed two simple, but in practice effective bounds. The POMDP upper bound bounds the objective function of the programs from above by the value of the optimal POMDP policy. We have defined a linear program that computes this policy and its value. In the next chapter, we shall see the extent of improvement in the performance of the 0-1 MILPs caused by these heuristics in solving sample DEC-POMDPs.

Chapter 7

Computational Experience

7.1 Introduction

In Chapters 3, 4 and 5, we have presented different mathematical programs for finding an optimal finite horizon joint policy of a given DEC-POMDP. These are listed in the following table. n is an integer ≥ 2.

Chapter	Program	Type	Label	Comments
3	(3.20)-(3.23)	Nonlinear Program	**NLP1**	Locally Optimal, n agents
4	(4.58)-(4.65)	0-1 MILP	**MILP1**	Optimal, n agents
4	(4.68)-(4.75)	0-1 MILP	**MILP2**	Optimal, n agents
5	(5.68)-(5.79)	0-1 MILP	**MILP3**	Optimal, 2 agents
5	(5.90)-(5.104)	0-1 MILP	**MILP4**	Optimal, n agents
5	(5.110)-(5.124)	0-1 MILP	**MILP5**	Optimal, n agents

The comments column mentions what sort of joint policy the program is *guaranteed* to find and for how many agents. In the remainder of the chapter, we shall first compare the sizes of these programs and then present the computational experience of these programs on sample DEC-POMDPs problems.

7.2 Comparison Of The Sizes Of Programs

The size of a program is measured in terms of the number of variables and constraints it consists of. The size of a program determines not only the space required to formulate the program in memory but also the time taken by an appropriate algorithm to solve it. For 0-1 MILPs, the number of 0-1 variables is also an important, separate measure of size since the time taken to solve a 0-1 MILP using the BB method is a function of this number. The sizes of these programs are as follows. Note that T is the horizon and n is the number of agents. 'Exp.' is for 'exponential in'.

Program	# Variables	# Constraints	# 0-1 Variables
NLP1	Exp. in T	Exp. in T	0
MILP1	Exp. in T and n	Exp. in T and n	Exp. in T
MILP2	Exp. in T and n	Exp. in T	Exp. in T
MILP3	Exp. in T	Exp. in T	Exp. in T
MILP4	Exp. in T and n	Exp. in T	Exp. in T
MILP5	Exp. in T and n	Exp. in T	Exp. in T

Program	# Variables	# Constraints
NLP1	$\sum_{i\in I}\|\mathcal{H}_i\|$	$\sum_{i\in I}\|\mathcal{I}_i\|$
MILP1	$\sum_{i\in I}\|\mathcal{H}_i\| + \prod_{i\in I}\|\mathcal{E}_i\|$	$\sum_{i\in I}\|\mathcal{I}_i\| + \prod_{i\in I}\|\mathcal{E}_i\|$
MILP2	$\sum_{i\in I}\|\mathcal{H}_i\| + \prod_{i\in I}\|\mathcal{E}_i\|$	$\sum_{i\in I}\|\mathcal{I}_i\| + \sum_{i\in I}\|\mathcal{E}_i\|$
MILP3	$3\sum_{i\in I}\|\mathcal{H}_i\| + \sum_{i\in I}\|\mathcal{I}_i\|$	$\sum_{i\in I}\|\mathcal{I}_i\| + 3\sum_{i\in I}\|\mathcal{H}_i\|$
MILP4	$3\sum_{i\in I}\|\mathcal{H}_i\| + \sum_{i\in I}\|\mathcal{I}_i\| + \prod_{i\in I}\|\mathcal{E}_i\|$	$\sum_{i\in I}\|\mathcal{I}_i\| + 3\sum_{i\in I}\|\mathcal{H}_i\| + \sum_{i\in I}\|\mathcal{E}_i\|$
MILP5	$3\sum_{i\in I}\|\mathcal{H}_i\| + \sum_{i\in I}\|\mathcal{I}_i\| + \sum_{i\in I}\|\mathcal{E}_{-i}\|$	$\sum_{i\in I}\|\mathcal{I}_i\| + 3\sum_{i\in I}\|\mathcal{H}_i\| + \sum_{i\in I}\sum_{k\in I\setminus\{i\}}\|\mathcal{E}_k\|$

Table 7.1: Sizes Of Different Mathematical Programs.

Program	# 0-1 Variables
MILP1	$\sum_{i\in I}\|\mathcal{E}_i\|$
MILP2	$\sum_{i\in I}\|\mathcal{E}_i\|$
MILP3	$2\sum_{i\in I}\|\mathcal{H}_i\|$
MILP4	$2\sum_{i\in I}\|\mathcal{H}_i\|$
MILP5	$2\sum_{i\in I}\|\mathcal{H}_i\| + \sum_{i\in I}\|\mathcal{E}_{-i}\|$

Table 7.2: Number Of 0-1 Variables In Different 0-1 MILPs.

A more detailed description of the sizes is given in Tables (7.1) and (7.2). Note that the number of constraints does not include **domain constraints**. A domain constraint is a constraint that explicitly determines the domain of a variable. Examples of domain constraints include,

$$\begin{aligned} x_i(h) &\geq 0 \\ y_i(\iota) &\in [-\infty, +\infty] \\ x_i(h) &\in \{0,1\} \\ z(j) &\in [0,1] \end{aligned}$$

We recall that for agent $i \in I$, $\mathcal{H}_i$ denotes the set of histories of lengths less than or equal to T of agent i and $\mathcal{I}_i$ denotes the set of information sets of agent i of lengths less than or equal to T - 1. $\mathcal{E}_i \subset \mathcal{H}_i$ denotes the set of histories of length T or terminal histories of agent i. The sizes of $\mathcal{H}_i$, $\mathcal{I}_i$ and $\mathcal{E}_i$ are exponential in T. To be precise,

$$|\mathcal{H}_i| = \sum_{t=1}^{T} |A_i|^t |O_i|^{t-1} \tag{7.1}$$

$$|\mathcal{I}_i| = \sum_{t=0}^{T-1} |A_i|^t |O_i|^t \tag{7.2}$$

$$|\mathcal{E}_i| = |A_i|^T |O_i|^{T-1} \tag{7.3}$$

Program	# Variables	# Constraints	# 0-1 Variables
NLP1	340	170	0
MILP1	16724	16554	256
MILP2	16724	426	256
MILP3	1190	1190	680
MILP4	17574	1446	680
MILP5	1446	1446	936

Table 7.3: Sizes Of The Programs For A 2-Agent, 2-Actions, 2-Observations, 4-Period Dec-POMDP.

Program	# Variables	# Constraints	# 0-1 Variables
NLP1	510	255	0
MILP1	$>$ 2 million	$>$ 2 million	384
MILP2	$>$ 2 million	639	384
MILP4	$>$ 2 million	2169	1020
MILP5	50937	2553	1788

Table 7.4: Sizes Of The Programs For A 3-Agent, 2-Actions, 2-Observations, 4-Period Dec-POMDP.

7.2.1 Examples

Concrete examples shall enable us to compare the sizes of these programs. For a 2-agent DEC-POMDP in which each agent has 2 actions and 2 observations and with $T = 4$, for each agent i, $|\mathcal{H}_i| = 170$, $|\mathcal{I}_i| = 85$ and $|\mathcal{E}_i| = 128$. The sizes of the programs for this example to find an optimal 4-period joint policy are given in Table (7.3). As the table shows and as discussed previously, for a 2-agent DEC-POMDP, the **MILP3** is the smallest program, but has more 0-1 variables than other programs which are overall larger than it.

All of the above programs except **MILP3** are capable of finding an optimal joint policy even when the number of agents is more than 2. For the example given above, if we add a third agent also with 2 actions and 2 observations, the sizes of the programs are as given in Table (7.4). Thus, we see that for a 3-agent DEC-POMDP, only **MILP5** is within a reasonable size to be formulated in memory. Note that the number of 4-period policies in the canonical form per agent for the the two examples is 2^{15}. This means that in the 2-agent case, the number of 4-period joint policies in the canonical form is thereby 2^{30} and in the 3-agent case, the number of 4-period joint policies in the canonical form is an astronomical 2^{45}. These figures are representative of the resources in terms of memory and time required by existing algorithms to find an optimal joint policy.

7.2.2 Summary Of The Comparison

We can summarize the tables presented above as follows.

- **NLP1** is the smallest program of all the programs. It is a nonconvex nonlinear program and therefore does not come with any guarantee of finding an optimal joint policy.

- **NLP1** can be converted to a 0-1 MILP upon soling which we are guaranteed to find an optimal joint policy. The five 0-1 MILPs **MILP1** to **MILP5** represent different ways of converting **NLP1** to a 0-1 MILP.
- For the 2-agent case, **MILP3** is the smallest mathematical program that finds an optimal joint policy. Notice that is only slightly larger than **NLP1**. However, it requires more 0-1 variables than **MILP1** and **MILP2**.
- For the 2-agent case, **MILP2** is the smallest program in terms of the number of 0-1 variables that finds an optimal joint policy. **MILP1** has the same number of 0-1 variables as **MILP2**, but since it is overall considerably larger than **MILP2**, the latter is considered as the smallest.
- For the 3-or-more agents case, **MILP5** is the smallest mathematical program. However it requires more 0-1 variables than **MILP2**, the next smallest mathematical program for this case. This means that in choosing **MILP5** over **MILP2** for solving a 3-agent DEC-POMDP, there is a potential trade-off between the time taken to find an optimal joint policy and the space required to find it.

7.3 Experimental Set-Up

In this section, we describe the manner in which we have tested the programs described above. We have tested the programs on five problems modeled as DEC-POMDPs.

- *The multi-agent (MA-Tiger) problem* [NTY+03]. This problem is modeled as a 2-agent DEC-POMDP with 2 states, 3 actions and 2 observations per agent, and solved up to horizon 4.
- *The multi-access broadcast channel (MABC) problem* [HBZ04]. The problem is modeled as a 2-agent DEC-POMDP with 4 states, 2 actions per agent and 2 observations per agent, and solved up to horizon 5.
- *Grid meeting problem* [Ber05]. The problem is modeled as a 2-agent DEC-POMDP with 16 states, 5 actions per agent and 2 observations per agent, and solved up to horizon 3.
- *Fire fighting problem* [OSV08]. The problem is modeled as a 2-agent DEC-POMDP with 27 states, 3 actions per agent and 2 observations per agent, and solved up to horizon 4.
- *A random problem.* The problem is modeled as a 2/3-agent DEC-POMDP with 50 states and with either 2 actions/observations per agent (akin to the MACB problem) or with 3 actions per agent and 2 observations per agent, similar to the MA-Tiger problem. The state transition function, the joint observation function, the reward functions and the initial state are generated randomly. For the 2-agent case, this problem is solved up to horizon 4. For the 3-agent case, this problem is solved up to horizon 3.

NLP1 was tested on all the problems except the random problem. Of the five candidate MILP programs, we have tested with mainly with **MILP2**, **MILP3** and **MILP5**. **MILP2** and **MILP3** were tested on all the problems. **MILP2** and **MILP5** were also tested on the 3-agent random problem. The other two programs could not be tested because they were found to be too large to be formulated in memory. Note that **MILP5** was not tested on 2-agent problems because in the 2-agent case, **MILP5** reduces to **MILP3**.

The MILPs were coded in the Java programming language. They were solved using the 0-1 MILP solver provided in the commercially available ILOG-CPLEX 10 software. The 'solver' of the software is a set of packages in the Java language which enable the formulation of an MILP (as well as of an ILP or an LP) and which contain the branch and bound method and the simplex algorithm. The computer on which **MILP2** and **MILP3** were formulated and solved was an Intel P4 machine with 3.40 gigahertz processor speed and 2.0 GB ram.

The NLP was solved using ready-to-use solvers available from the NEOS server. We tested on three different solvers: SNOPT, LANCELOT and LOQO.

The programs were tested in their original versions (as presented in Chapters 4 and 5) as well as with the application of the heuristics described in Chapter 6. The notation used for the heuristics employed is as follows. Let **M** denote **MILP2** or **MILP3**.

- **M** without variables for locally extraneous histories shall be denoted by **M**-LOC.
- **M** without variables for globally extraneous histories shall be denoted by **M**-GLOB.
- **M** with the addition of the upper bound POMDP bound cut shall be denoted by **M**-Up.
- **M** with the addition of the lower bound DEC-POMDP cut shall be denoted by **M**-Low.

We have also used combinations of heuristics. Thus, **M**-LOC-Up means that **M** was formulated without variables for locally extraneous histories and the lower bound DEC-POMDP cut was added.

Note that we have also tested **NLP1** on the the MA-Tiger and the MABC problems. **NLP1** was coded in the AMPL language and solved on the NEOS solver SNOPT. MA-Tiger was tested for horizons 2 and 3 and MABC was tested for horizons 3 and 4. In both cases, the joint policy found was sub-optimal. The time taken to solve the problems was of an order of a minute.

7.3.1 Measurement Of Time Taken

In the following three sections, we present the computational experience of the NLP and the two MILPs on the five problems. The tables presented in the following sections list the times taken by the two programs in solving the three problems. Each entry in the table shows the time taken in seconds by a program **M** to solve a problem P. The time taken shown is the total time taken to find an optimal joint policy. In order words, the time taken shown in the table is a *sum* of the following times:

1. The time taken to compute the value $\mathcal{R}(\alpha, j)$ and joint observations sequence probability $\Psi(\alpha, j)$ of each terminal joint history $j \in \mathcal{E}$.
2. If one or more heuristic is used, the time taken to compute the quantities required by the concerned heuristics:
   (a) For LOC, the time taken to identify every locally extraneous history of each agent. This means the time taken to run the procedure described in Section (6.3.2) of Chapter 6.

(b) For GLOB, the time taken to identify every locally extraneous history of each agent. This means the time taken to run the procedure described in Section (6.5.1) of Chapter 6.

(c) For Up, the time taken to find an optimal T-period POMDP policy. This means the time taken to set up and solve the linear program **LP4** given in Section (6.7.1) of Chapter 6.

(d) For Low, the time taken to find an optimal (T-1)-period DEC-POMDP joint policy for P. This means the time taken to set up and solve **M** for P for this horizon.

3. The time taken to set up and solve **M** using the BB method.

7.4 The MA-Tiger Problem

The multi-agent tiger (MA-Tiger) problem is adapted from the single-agent tiger problem which can be modeled as a POMDP. The problem is described as follows.

We are given two persons confronted with two closed doors. Behind one door is a tiger, behind the other an escape route. The persons do not know which door leads to what. If either of them opens the wrong door, the lives of both will be imperiled. If they both open the right door, they will be free. The persons have a limited time in which to decide which door to open. They can use this time to gather information about the precise location of the tiger.

The persons can gather information by hearing for the noises that tiger makes from behind the doors. However, in addition to the noises made by the tiger, other noises may emanate from behind the doors as well. So, upon hearing a noise, a person is not sure that it is the tiger who has made the noise. Moreover, a noise emanating from either door is not necessarily heard by both persons. Finally, the persons are unable to share information with one another. So, at any time, a person cannot tell the other person if at that time step he heard a noise or not.

As stated, the persons have a limited amount of time before deciding which door to open. We imagine time to be split into discrete time periods. Our objective is to model this situation as a finite horizon DEC-POMDP. An optimal joint policy for such a DEC-POMDP would tell each person what to do in each period: open a door or listen for noises.

In modeling this problem as a DEC-POMDP, we obtain a 2-agent, 2-state, 3-actions-per-agent, 2-observations-per agent DEC-POMDP whose elements are as follows.

- Each person is an agent. So, we have a 2-agent DEC-POMDP.
- The state of the problem is described by the location of the tiger. Thus, S consists of two states Left (tiger is behind the left door) and Right (tiger is behind the right door).
- Each agent's set of actions consists of three actions: Open Left (open the left door), Open Right (open the right door) and Listen (listen).
- Each agent's set of observations consists of two observations: Noise Left (noise coming from the left door) and Noise Right (noise coming from the right door).

Joint Action	State	Joint Observation	Probability
(Listen, Listen)	Left	(Noise Left, Noise Left)	0.7225
(Listen, Listen)	Left	(Noise Left, Noise Right)	0.1275
(Listen, Listen)	Left	(Noise Right, Noise Left)	0.1275
(Listen, Listen)	Left	(Noise Right, Noise Right)	0.0225
(Listen, Listen)	Right	(Noise Left, Noise Left)	0.0225
(Listen, Listen)	Right	(Noise Left, Noise Right)	0.1275
(Listen, Listen)	Right	(Noise Right, Noise Left)	0.1275
(Listen, Listen)	Right	(Noise Right, Noise Right)	0.7225
(*, *)	*	(*, *)	0.25

Table 7.5: Joint Observation Function $\mathbb{G}$ For The MA-Tiger Problem.

Joint Action	Left	Right
(Open Right, Open Right)	20	-50
(Open Left, Open Left)	-50	20
(Open Right, Open Left)	-100	-100
(Open Left, Open Right)	-100	-100
(Listen, Listen)	-2	-2
(Listen, Open Right)	9	-101
(Open Right, Listen)	9	-101
(Listen, Open Left)	-101	9
(Open Left, Listen)	-101	9

Table 7.6: Reward Function A For The MA-Tiger Problem.

The initial state is an equi-probability distribution over S. The state transition function $\mathbb{P}$, joint observation function $\mathbb{G}$ and the reward function R are taken from [NTY$^+$03]. $\mathbb{P}$ is quite simple. If one or both agents opens a door in a period, the state of the problem in the next period is set back to α. If both agents listen in a period, the state of the process in unchanged in the next period. $\mathbb{G}$ is also quite simple, and is given in Table (7.5). [NTY$^+$03] describes two reward functions called A and B for this problem. Reward function A is given in Table (7.6). Reward function B is identical to A with the exception that the joint action (Open Right, Open Right) gives a reward of 0 when the state is Right. The value of the optimal joint policy for this problem for different horizons in given in Table (7.7).

Reward Function	Horizon T	Value Of Optimal Joint Policy
A	3	5.19
	4	4.80
B	3	30
	4	40

Table 7.7: Value Of An Optimal Joint Policy For The MA-Tiger Problem.

Reward Function	Program	Horizon T	Time (Secs)	Horizon T	Time (Secs)
A	**MILP2**	3	3.7	4	*
	MILP2-Low	3	4.9	4	72
	MILP2-Up	3	3.5	4	*
	MILP2-LOC	3	6.4	4	*
	MILP2-LOC-Low	3	7.6	4	175
	MILP2-LOC-Up	3	6.2	4	*
B	**MILP2**	3	0.95	4	*
	MILP2-Low	3	1.0	4	43
	MILP2-Up	3	1.6	4	*
	MILP2-LOC	3	3.6	4	*
	MILP2-LOC-Low	3	3.7	4	146
	MILP2-LOC-Up	3	4.3	4	*

Table 7.8: Times Taken By **MILP2** On The MA-Tiger Problem.

Reward Function	Program	Horizon T	Time (Secs)	Horizon T	Time (Secs)
A	**MILP3**	3	11.16	4	*
B	**MILP3**	3	12.33	4	*

Table 7.9: Times Taken By **MILP3** On The MA-Tiger Problem.

Table (7.8) lists the times taken by **MILP2** (with and without different heuristics) on the MA-Tiger problem. Table (7.9) lists the times taken by **MILP3** on the MA-Tiger problem. '*' denotes that a time-out occurred where time-out was set to 30 minutes. The time out did not occur in the computation of the concerned heuristic, but in the solving of the program by the 0-1 MILP solver. Note that *none* of the histories of any agent was found to be locally extraneous for the given initial state in the MA-Tiger problem.

Table (7.10) gives the times taken by the different existing exact and inexact algorithms (described in Chapter 2) on the MA-Tiger problem for different horizons. The times for the algorithms were taken from the papers in which they were presented. In the table, E denotes an exact algorithm, N denotes an algorithm that finds a Nash Equilibrium and A denotes an approximate algorithm. * indicates a time out (although it is not reported, it is safe to assume that the time out is probably of a few hours).

From these tables, we see that finding an optimal joint policy for the MA-Tiger problem by solving **MILP2** or **MILP3** is indeed much faster than by using existing algorithms. Of the two, **MILP2** is faster. For horizon 3, for reward function A, the best time obtained is 3.5 secs, by **MILP2**-Up and for reward function B, the best time obtained is 0.95 secs, by **MILP2** (without the aid of any heuristics). For horizon 4, for reward function A, the best time obtained is 72 secs, by **MILP2**-Low and for reward function B, the best time obtained is 43 secs, also by **MILP2**-Low. Note that **MILP3** is unable to solve the MA-Tiger problem for horizon 4 in a reasonable amount of time. Heuristics do not seem to help in this regard either.

Reward Function	Algorithm	Type	Horizon	Time	Horizon	Time
A	MAA*	E	3	4 s	4	$>$ 1 month
	Recursive MAA*	E	3	4 s	4	2.5 h
	Exhaustive-JESP	N	3	317 s	4	*
	DP-JESP	N	3	0	4	0.02 s
	MBDP	A	3	0.19 s	4	0.46 s
B	MAA*	E	3	1s	4	25 h
	Recursive MAA*	E	3	1s	4	25 h

Table 7.10: Times Taken By Existing Algorithms On The MA-Tiger Problem.

These results show that the reward function can determine the time taken in finding an optimal joint policy. Existing algorithms also show a sensitivity to the reward function apropos the time taken to find an optimal joint policy.

Note that even the inexact algorithms listed in Table (7.10) find an optimal joint policy for the MA-Tiger problem, reward function A. It is probably some feature of the reward function and the dynamics of the MA-tiger problem that allows them to do so. However, for reward function B, some algorithms, such as the JESP, are not able to find an optimal joint policy.

7.5 The MABC Problem

We described this problem in Chapter 1 following [Ros83]. However, that description only allows us to model the problem as a transition independent DEC-MDP. [HBZ04] describe it slightly differently which allows us to model it as a DEC-POMDP. This description is closer to the one given in [OW96] and is as follows.

We are given two nodes (computers) who are required to send messages to each other over a common channel for a given duration of time. Time is imagined to be split into discrete periods. Each node has a buffer with a capacity of one message. A buffer that is empty in a period is refilled with a certain probability in the next period. In a period, only one node can send a message. If both send a message in the same period, a collision of the messages occurs and neither message is transmitted. In case of a collision, each node is intimated about it through a collision signal. But the collision signaling mechanism is faulty. In case of a collision, with a certain probability, it does not send a signal to either one or both nodes.

We are interested in pre-allocating the channel amongst the two nodes for a given number of periods. The pre-allocation consists of giving the channel to one or both nodes in a period as a function of the node's information in that period. A node's information in a period consists only of the sequence of collision signals it has received till that period.

In modeling this problem as a DEC-POMDP, we obtain a 2-agent, 4-state, 2-actions-per-agent, 2-observations-per agent DEC-POMDP whose elements are as follows.

- Each node is an agent.

Horizon T	Value Of Optimal Joint Policy
3	2.99
4	3.89
5	4.79

Table 7.11: Value Of An Optimal Joint Policy For The MABC Problem.

- The state of the problem is described by the states of the buffers of the two nodes. The state of a buffer is either Empty or Full. Hence, the problem has four states, (Empty, Empty), (Empty, Full), (Full, Empty) and (Full, Full).
- Each node has two possible actions, Use Channel and Don't Use Channel.
- In a period, a node may either receive a collision signal or it may not. So each node has two possible observations, Collision and No Collision.

The initial state of the problem α is (Full, Full). The state transition function $\mathbb{P}$, the joint observation function $\mathbb{G}$ and the reward function R have been taken from [HBZ04]. If both agents have full buffers in a period, and both use the channel in that period, the state of the problem is unchanged in the next period; both agents have full buffers in the next period. If an agent has a full buffer in a period and only he uses the channel in that period, then his buffer is refilled with a certain probability in the next period. For agent 1, this probability is 0.9 and for agent 2, this probability is 0.1. If both agents have empty buffers in a period, irrespective of the actions they take in that period, their buffers get refilled with probabilities 0.9 (for agent 1) and 0.1 (for agent 2).

$\mathbb{G}$ is as follows. If the state in a period is (Full, Full) and the joint action taken by the agents in the previous period is (Use Channel, Use Channel), the probability that both receive a collision signal is 0.81, the probability that only one of them receives a collision signal is 0.09 and the probability that neither of them receives a collision signal is 0.01. For any other state the problem may be in a period and for any other joint action the agents may have taken in the previous period, the agents do not receive a collision signal.

R is quite simple. If the state in a period is (Full, Empty) and the joint action taken is (Use Channel, Don't Use Channel) or if the state in a period is (Empty, Full) and the joint action taken is (Don't Use Channel, Use Channel), the reward is 1; for any other combination of state and joint action, the reward is 0.

The value of the optimal joint policy for this problem is given in Table (7.11).

Table (7.12) lists the times taken by **MILP2** with and without heuristics on the MABC problem. Table (7.13) lists the times taken on the MABC problem. '-' denotes that the program could not be formulated in memory. '*' indicates time out of 30 minutes.

Table (7.14) gives the times taken by the different existing exact and inexact algorithms on the MABC problem for different horizons. Again, the times for the algorithms were taken from

Program	Horizon T	Time (Secs)	Horizon T	Time (Secs)	Horizon T	Time (secs)
MILP2	3	0.86	4	900	5	-
MILP2-Low	3	0.93	4	900	5	-
MILP2-Up	3	1.03	4	907	5	-
MILP2-LOC	3	0.84	4	80	5	*
MILP2-LOC-Low	3	0.84	4	120	5	*
MILP2-LOC-Up	3	0.93	4	10.2	5	25

Table 7.12: Times Taken By **MILP2** On The MABC Problem.

Program	Horizon T	Time (Secs)	Horizon T	Time (Secs)	Horizon T	Time (secs)
MILP3	3	0.391	4	3.53	5	-

Table 7.13: Times Taken By **MILP3** On The MABC Problem.

the papers in which they were presented. Blanks indicate that the algorithm either ran out of time or space.

Again, as the tables show, finding an optimal joint policy for the MABC problem by solving **MILP2** or **MILP3** requires much less time than using existing algorithms. For horizons 3 and 4, for the MABC, **MILP3** is much faster than **MILP2**. However, **MILP3** is unable to solve for horizon 5; it incurs a time-out. Note that the MABC problem can be considered an "easy" problem compared to the MA-Tiger problem for the following two reasons:

- About 62% of all histories in the MABC problem are locally extraneous, whereas, as stated before, 0% of histories in the MA-Tiger problem are locally extraneous.
- The MABC problem is smaller than the MA-Tiger problem since it has only 2 actions per agent, whereas the latter has 3.
- The value of the optimal T-period POMDP policy equals the value of the optimal T-period joint policy for the MABC problem; for the MA-Tiger problem, the latter has a much lower value. This means that upper bound POMDP cut comes into play in the MABC problem (it reduces the time taken by the BB method), but it has no effect in the MA-Tiger problem.

Algorithm	Type	Horizon	Time	Horizon	Time	Horizon	Time
DP	E	3	5 s	4	900 s	5	
MAA*	E	3	< 1 s	4	3 h	5	
Recursive MAA*	E	3	< 1 s	4	1.5 h	5	
PBDP	E	3	< 1 s	4	2 s	5	10^5 s
Approx. PBDP	A	3	< 1 s	4	< 1 s	5	10 s
MBDP	A	3	0.01	4	0.01	5	0.02 s

Table 7.14: Times Taken By Existing Algorithms On The MABC Problem.

Program	Least Time (secs)	Most Time (secs)	Average	Std. Deviation
MILP2	2.45	455	120.6	183.48
MILP3	6.85	356	86.88	111.56

Table 7.15: Times Taken By **MILP2** and **MILP3** On the 2-Agent Random1 Problem For Horizon 4.

Program	Least Time (secs)	Most Time (secs)	Average	Std. Deviation
MILP2	1.45	10.46	4.95	3.98
MILP3	5.06	12.53	7.28	2.43

Table 7.16: Times Taken By **MILP2** and **MILP3** On the 2-Agent Random2 Problem For Horizon 3.

7.6 Random Problem

The random problem consists of solving a randomly generated 2 or 3-agent DEC-POMDP. In a randomly generated DEC-POMDP, the state transition function, the joint observation function and the reward functions are randomly generated. We tested the programs on two different sized random problems, which we shall call **Random1** and **Random2**. Random1 has 2 actions and 2 observations per agent while Random3 has 3 actions and 2 observations per agent. Thus, Random1 has the same size as the MABC problem while Random2 has the same size as the MA-Tiger problem. The number of states in each problem is 50. Rewards were randomly generated integers in the range 1 to 5.

MILP2 and **MILP3** were tested on the 2-agent Random1 problem and the 2-agent Random2 problem. **MILP2** was also tested on the 3-agent Random1 problem.

Table (7.15) shows the times taken by **MILP2** and **MILP3** on the 2-agent Random1 problem for horizon 4. Table (7.16) shows the times taken by **MILP2** and **MILP3** on the 2-agent Random2 problem for horizon 3. Finally, Table (7.17) shows the times taken by **MILP2** and **MILP5** to solve the 3-agent Random1 problem for horizon 3. The times in all the three tables were averaged over 10 runs.

As described in Section (7.2), on any given (2-agent) problem, **MILP3** has far fewer variables and constraints **MILP2**. However, on any given (2-agent problem), the number of 0-1 variables in **MILP3** is more than the number of 0-1 variables in **MILP2**. In the Random2 problem, the larger number of 0-1 variables slows down **MILP3** whereas in the Random1 problem, the number of 0-1 variables is small enough so that overall smallness of **MILP3** allows the BB method to solve it faster than **MILP2**.

7.7 Experience of the NLP

Descriptions of the two problems, Grid Meeting and Fire-fighting can be obtained from [Ber05] and [OSV08] respectively. The comparative computational experience of the three programs

Program	Least Time (secs)	Most Time (secs)	Average	Std. Deviation
MILP2	21	173	70.6	64.02
MILP2-Low	26	90	53.2	24.2
MILP5	754	2013	1173	715

Table 7.17: Times Taken By **MILP2** and **MILP5** On the 3-Agent Random1 Problem For Horizon 3.

NLP1, **MILP2** and **MILP3** on the four problems MABC, MA-Tiger, Grid Meeting and Fire-Fighting is given the following four subsections. Note that the existing algorithms to which we compared our programs are: GMAA* [OSV08], PBDP [SC06], DP-LPC [BCd08], DP [HBZ04] and MAA* [SCZ05].

7.7.1 MABC

Horizon	Algorithme	Solver	Valeur	Temps (s)
4	GMAA*	-	3.89	0.03
	PBDP	-	3.89	2.00
	DP-LPC	-	3.89	4.59
	DP	-	3.89	17.59
	MAA*	-	3.89	$\mathcal{O}(10^4)$
	NLP1	SNOPT	3.17	0.01
	NLP1	LANCELOT	3.79	0.95
	NLP1	LOQO	3.79	0.05
	MILP2	ILOG	3.89	10.2
	MILP3	ILOG	3.89	3.53
5	GMAA*	-	4.79	5.68
	PBDP	-	4.79	$\mathcal{O}(10^5)$
	NLP1	SNOPT	4.70	0.21
	NLP1	LANCELOT	4.69	20.00
	NLP1	LOQO	4.69	0.18
	MILP2	ILOG	4.79	25.00
	MILP3	ILOG	4.79	$\mathcal{O}(10^3)$

7.7.2 MA-Tiger

Horizon	Algorithme	Solver	Valeur	Temps (s)
3	GMAA*	-	5.19	0.04
	DP-LPC	-	5.19	1.79
	DP	-	5.19	2.29
	MAA*	-	5.19	4.00
	NLP1	SNOPT	-45	0.03
	NLP1	LANCELOT	5.19	0.47
	NLP1	LOQO	5.19	0.016
	MILP2	ILOG	5.19	3.17
	MILP3	ILOG	5.19	11.16
4	GMAA*	-	4.80	3209
	DP-LPC	-	4.80	535
	NLP1	SNOPT	-9.80	4.62
	NLP1	LANCELOT	4.80	514
	NLP1	LOQO	4.78	91
	MILP2	ILOG	4.80	72
	MILP3	ILOG	4.80	$\mathcal{O}(10^3)$

7.7.3 Grid Meeting

Horizon	Algorithme	Solver	Valeur	Temps (s)
2	GMAA*	-	0.91	0
	NLP1	SNOPT	0.91	0.01
	NLP1	LANCELOT	0.91	0.06
	NLP1	LOQO	0.91	0.076
	MILP2	ILOG	0.91	0.65
	MILP3	ILOG	0.91	0.61
3	GMAA*	-	1.55	5.81
	NLP1	SNOPT	1.55	1.05
	NLP1	LANCELOT	1.55	257
	NLP1	LOQO	0.48	81
	MILP2	ILOG	1.55	1624
	MILP3	ILOG	1.55	$\mathcal{O}(10^3)$

7.7.4 Fire Fighting

Horizon	Algorithme	Solver	Valeur	Temps (s)
3	GMAA*	-	-5.73	0.41
	NLP1	SNOPT	-5.73	0.05
	NLP1	LANCELOT	-5.73	2.49
	NLP1	LOQO	-5.80	0.24
	MILP2	ILOG	-5.73	$\mathcal{O}(10^3)$
	MILP3	ILOG	-5.73	38
4	GMAA*	-	-6.57	5510
	NLP1	SNOPT	-6.57	4.61
	NLP1	LANCELOT	-6.62	1637
	NLP1	LOQO	-6.64	83
	MILP2	ILOG	-6.57	$\mathcal{O}(10^3)$
	MILP3	ILOG	-6.57	$\mathcal{O}(10^3)$

7.8 Summary

In this chapter we have compared and tested the 0-1 MILPs presented in Chapters 4 and 5 on the MA-Tiger and the MABC problem, as well as on random problems. Computational experience shows that these programs indeed find an optimal joint policy in much less time than existing algorithms. The experience also shows the important role heuristics play in reducing the time taken to find an optimal joint policy. Eventhough **MILP3** is the smallest of the five programs (i.e., has the least number of variables and constraints), it is not able to solve certain problems which the much bigger **MILP2** is able to. This may be due to the fact that **MILP3** has more 0-1 variables than **MILP2**.

In the next chapter, we present our conclusions and point to ways in which our approach can be extended to solve larger DEC-POMDPs.

Chapter 8

Conclusions And Future Work

8.1 Conclusions

Our thesis has studied the problem of planning for decentralized problems. Such problems arise when one attempts to automatically control a process through several independent seats of control. Such decentralized control is necessary in many applications and desirable in many others. Examples of such applications include the decentralized detection problem [TA85] from the domain of communication networks, the multi-access broadcast channel problem [Ros83] from the domain of operations research, the multi-rover exploration problem [BZLG04] from the domain of robotics, etc.

But planning for decentralized problems is computationally very difficult. This high computational difficulty has been formally recognized for at least the last two decades. In 1985, Tsitsiklis and Athans [TA85] proved that the team decision problem (TDP), a mathematical model for static decentralized problems, proposed in 1959 by Radner [Rad59], is NP-hard. The general case, of dynamic decentralized problems, is captured by the decentralized partially observable Markov decision process or DEC-POMDP model or equivalently, by the Markov Team Decision Problem (MTDP) model, and Bernstein and collaborators proved in 2002 that dynamic decentralized problems are even harder; they are NEXP-complete [BGIZ02].

These negative complexity results have in part deterred the development of algorithms for planning for decentralized problems, particularly those of a dynamical nature. However, with specular advances in computational power over the past decade and more, we are at a stage where we are capable of solving at least small instances of decentralized problems using a reasonable amount of time and resources, and therefore there has been a renewed interest over the past few years in the conception of such algorithms. Papers published on this subject since 2002 for example - [BGIZ02], [CSC02], [NTY+03], [BZLG04], [CRRL04], [SCZ05], [BM06], [SC06], [PZ07a], [ABZ07], [SZ07] etc. - contain a wide variety of new techniques and algorithms for decentralized problems.

However, existing exact algorithms for DEC-POMDPs still seem to exhibit worst-case behavior on even small instances. In theory, they either require space that is doubly exponential in the duration (horizon) of the problem and/or time that is doubly exponential in the horizon, and this seems to be so even in practice.

Our thesis has sought to contribute to this growing body of work by proposing algorithms that are capable of solving small instances of DEC-POMDPs using relatively (compared to algorithms that exist) little space and time. The algorithms presented in this thesis demonstrate a markedly superior runtime over those exhibited by existing algorithms. Moreover, the worst-case space requirement of our algorithms is also only exponential in the horizon, and not doubly exponential in it.

There are two relevant byproducts of our approach for DEC-POMDPs. The first is that our approach can be used to find a finite horizon policy (in the sequence form) for a POMDP. The linear program **LP4** given in Chapter 6 finds such a policy. The second is that our approach yields an algorithm for finding a sample Nash Equilibrium for imperfect information 2-agent extensive games or 2-agent POSGs as well as another algorithm for finding a sample *pure* Nash equilibrium for imperfect information n-agent extensive games or n-agent POSGs, $n > 2$, provided that the game has a pure Nash Equilibrium. Currently, there are few algorithms for the latter class of games.

The main difference between our approach and existing ones is that we use the *sequence form* of the policy rather than the canonical tree form of the policy. The theoretical and intuitive advantage of our approach was highlighted in Chapter 3 (especially in its summary) and its practical advantage is borne out by the computational experience presented in Chapter 7. Our approach consists of different 0-1 mixed integer linear program (MILP) formulations each of whose solutions is an optimal joint policy. Therefore, a key component in the success of our approach is the robustness, efficiency and versatility of existing 0-1 MILP solvers (we have used the solver provided by the vendor ILOG, but one can safely assume that other, equally robust (and even freely available, such as those of the NEOS platform) solvers exist). The robustness of existing 0-1 MILP solvers is no accident. A variety of practical optimization problems, especially those of resource allocation, can be modeled as 0-1 MILPs, and therefore there is a natural demand, in industry and in the applied sciences, for such solvers. Our approach has thus profited from the advances made in linear programming techniques.

The computational experience of our approach reveals three things.

- First, from this experience, it is amply clear that the time taken to find an optimal joint policy in the sequence form by solving a 0-1 MILP is lesser by a magnitude that is usually of an order or two than existing algorithms take to find an optimal joint policy in the canonical form.
- Second, heuristics play a rather important role in reducing the time taken to solve a 0-1 MILP. Indeed, one can argue that without these heuristics, our approach does not present an advantage over existing algorithms. This is of course being unfair to our approach since existing algorithms too use heuristics (in fact analogous, to the ones we employ) - in the absence of which they would take even longer to find an optimal joint policy than they do in practice.
- Third, while our approach is considerably faster than existing algorithms, like the existing algorithms, it is only able to solve small DEC-POMDPs (number of actions and observations per agent limited to 3 or less) for short horizons (5 or less) in reasonable time. In other words, our approach essentially only represents a major advance in reducing the time required to find an optimal joint policy for small DEC-POMDPs for short horizons.

It is possible that a small DEC-POMDP with a short horizon corresponds to some practical decentralized problem. Solving such problems exactly in quick time would be therefore desirable. Needless to say, our approach is ideal for solving such problems.

We may thus summarize that existing approaches (including the one presented in this thesis) are capable of exactly solving only small DEC-POMDPs for short horizons. By 'exactly solving', we mean find an optimal joint policy for. Most practical problems require a much larger number of actions, observations and states, and conceivably have much longer horizons. Chapter 7 shows clearly that large DEC-POMDPs (say with 10 actions and observations per agent) and/or longer horizons (say of the order of 10, 50 etc.) cannot be solved in an exact manner by existing approaches.

8.2 Directions For Future Work

We now present some ideas on adapting our approach to larger DEC-POMDPs. We shall discuss the following two adaptations:

- Long Horizon Problems.
- Infinite Horizon Problems.

8.2.1 Long Horizon Problems

To solve for long horizons, we are required to transformed our exact approach (that finds a short-horizon, optimal joint policy) to an approximate approach that finds a long-horizon, but (most certainly) a sub-optimal joint policy.

A simple, but (in some cases) effective approach to tackle long horizons is to simply break a long horizon into a series of short horizons, and solve exactly for each short horizon using any of the 0-1 MILPs we have proposed. This will give us a set of short horizon optimal joint policies which will together constitute a long horizon sub-optimal joint policy. Such an approach is viable because as seen from Chapter 7, a short-horizon optimal joint policy can be found using any of our 0-1 MILPs in very little time. Schematically, this approach is described as follows. We are given a very long horizon denoted by $\mathbb{T}$.

1. Set k to 1. Set β^k to α, the initial state of the DEC-POMDP.
2. Find an optimal T-period joint policy p^k in the sequence form for initial state β^k by solving a 0-1 MILP (T is assumed to be very small, say 3 or 4).
3. Determine a state $\beta' \in \Delta(S)$ that is reached when p^k is executed from β^k (the manner in which β' may be chosen is given below).
4. Increment k by 1, set β^k to β'. If k is greater than $\mathbb{T}/T$, then stop; otherwise to Step 2.

This approach will give us a $\mathbb{T}$-period sub-optimal joint policy composed of the $\mathbb{T}/T$ joint policies, p^1, p^2, ..., $p^{\mathbb{T}/T}$. The agents will use p^1 for the first T periods, p^2 for the next T periods and so on. Considering the fact that a 4-period optimal joint policy for the MA-Tiger problem can found in around 40 seconds by **MILP2**, a 100-period sub-optimal joint policy for the MA-Tiger problem using the above approach can be found in about a 1000 seconds.

The key step in this algorithm is evidently Step 3, and β' can be determined in several interesting ways. The execution of any T-period joint policy from a given initial state puts the DEC-POMDP in a state $\beta \in \Delta(S)$ from a set of possible states $\mathcal{B} \subset \Delta(S)$. $\mathcal{B}$ is determined by the joint policy and by the initial state. Thus, in Step 3, the set $\mathcal{B}$ generated by p^k when the initial state is β^k must be considered for determining β'. We suggest the following three ways:

- We may let β' to be the *most probable state* defined as follows. Each state $\beta \in \mathcal{B}$ also has a probability Prob.$(\beta|\beta^k, p^k)$ of being reached when p^k is executed from β^k. We may simply let β' to the state in $\mathcal{B}$ with the highest probability.
- We may let β' to be a *compound state* defined as, for each $s \in S$, $\beta'(s) = \sum_{\beta \in \mathcal{B}} \text{Prob.}(\beta|\beta^k, p^k)\beta(s)$.
- We may let β' to be the most *promising state* defined as follows. For each $\beta \in \mathcal{B}$, we find an optimal T-period joint policy using a 0-1 MILP; the β at which the optimal joint policy has the largest value may be chosen as β'.

This approach is no doubt naive, and we can easily construct counter-examples in which it would lead to arbitrarily sub-optimal joint policies, worse even than randomly chosen joint policies. However, in problems such as the MABC and the MA-Tiger, it does seem to produce better joint policies than random joint policies. We have conducted a few experiments with this approach for horizons upto a 100 (using the compound state in Step 3), and the value averaged over several runs of a joint policy found by this approach was certainly much larger that the value obtained by a random joint policy. In fact, the quality of the joint policy is comparable to that produced by the MBDP Algorithm.

Notice that this approach is not possible with existing exact algorithms because they take a very long time even to find a short-horizon optimal joint policy. In other words, the DP, MAA* or PBDP algorithms cannot be used in Step 2 instead of solving a 0-1 MILP to find an optimal T-period joint policy. Thus viability of this albeit naive approach rests on the rapidity of solving a 0-1 MILP.

Another way to scale to longer horizons is a resource-bounded approach. In Chapter 2, we have described in brief two approximate algorithms, the Approximate PBDP (A-PBDP) Algorithm [SC06] and the MBDP Algorithm [SZ07]. Each of these approximate algorithms has been transformed from an exact algorithm: the former from the PBDP algorithm [SC06] and the latter from an algorithm that combines the DP Algorithm [HBZ04] with the MAA* Algorithm [SCZ05]. The central idea of each of the approximate algorithms is simple: Fix an upper bound on the amount of space and time that may be used, and find an optimal joint policy for that upper bound. Exact algorithms, on the other hand, assume that space and time available for finding an optimal joint policy are unlimited.

Using this principle, A-PBDP is able to find (sub-optimal) joint policies for the MABC problem for horizons, 5, 6, 7 and 8, in reasonable time, a small improvement over PBDP which is able to solve the problem exactly for horizon 4. On the other hand, MBDP is able to find (sub-optimal) joint policies for the MABC problem for really long horizons, from 100 to 100,000. Of course, MBDP provides no bounds on the sub-optimality of the joint policy found. So, it is unclear how sub-optimal the joint policy found is, but it certainly does seem to be much better than a randomly generated joint policy (note that the approximate approach we discussed above

compares favorably with MBDP, at least on the MABC and the MA-Tiger problem in terms of the quality of the joint policy produced and the time taken to find it).

We can adopt the principle of limited space for the exact approach presented in this thesis. This would entail that we fix the size of the 0-1 MILP and find an optimal joint policy in the sequence form for the given size of the MILP. The size of a 0-1 MILP is a function of the number of histories of lengths less than or equal to the horizon, of each agent. Thus, for a given (long) horizon, our approximate approach would consist of limiting the number of histories of each agent that are represented in the 0-1 MILP. In other words, it would consist of limiting the number of variables and constraints in the MILP. We would then be required to conceive ways or heuristics in which histories for each agent are chosen so that with the limited number of histories, a joint policy with maximum value can be found.

8.2.2 Infinite Horizon Problems

The thesis has dealt exclusively with the resolution of the finite horizon DEC-POMDP problem described in Chapter 2. In this problem, we are given an integer T representing the duration or horizon of the problem, and the optimality criterion is to maximize the expected sum of rewards obtainable in T time periods.

An important class of decentralized problems are those in which T is not given. In such problems the horizon is considered to be infinite. In other words, the number of time periods is infinite. Two criteria for optimality are possible for infinite horizon problems (s^t denotes the state of the process in period t and a_i^t denotes the action taken by agent i in period t):

- *Discounted sum of rewards*: Here, we are required to maximize the expectation of the sum of discounted rewards obtainable over the infinite horizon for a discount factor $\gamma \in (0, 1)$:

$$E\Big\{ \sum_{t=1}^{\infty} \gamma^t R(s^t, (a_1^t, a_2^t, \ldots, a_n^t)) \Big\} \tag{8.1}$$

- *Average reward per period*: Here we are required to maximize the expected average reward per period obtainable over the infinite horizon:

$$E\Big\{ \lim_{T \to \infty} \sum_{t=1}^{T} R(s^t, (a_1^t, a_2^t, \ldots, a_n^t))/T \Big\} \tag{8.2}$$

In finite horizon DEC-POMDPs, an optimal joint policy (be it in the canonical or the sequence-form) is a *non stationary* joint policy; each policy in it is non stationary. A non stationary policy is a function of time periods. To be precise, a non stationary policy prescribes an action for every sequence of observations of lengths less than T receivable by the agent. For infinite horizon DEC-POMDPs, non stationary joint policies are not conceivable since they would require infinite space. A non stationary policy of an agent for an infinite horizon problem would be required to map every possible sequence of observations of every possible length receivable by the agent to an action. Therefore, for such problems only *stationary* policies are conceivable. A stationary policy is independent of the time period.

For infinite horizon DEC-POMDPs, an optimal (stationary) joint policy can be represented as a tuple of finite state machines called *controllers* [BHZ05]. The tuple of controllers is called a *joint controller*. A **deterministic controller** for agent $i \in I$ is described as the tuple (N_i, D_i, E_i), where N_i is a set of *nodes*, D_i is a function from N_i to A_i, and E_i is a function from $N_i \times O_i$ to N_i. The agent begins using the controller in the first period by selecting a node $z^1 \in N_i$, and taking the action $D_i(z^1)$. In the second period, the agent moves to the node $z^2 = E(z^1, o_i^2)$ where o_i^2 is the observation he receives in the second period. In the second period, he takes the action $D_i(z^2)$ and moves to the node $z^3 = E(z^2, o_i^3)$ in the third period, and so on perpetually. A **stochastic controller** is similar to a deterministic controller. However, D_i is a function from N_i to $\Delta(A_i)$, the set of probability distributions over A_i and E_i is a function from $N_i \times O_i$ to $\Delta(N_i)$, the set of probability distributions over N_i. So, in using a stochastic controller, an agent is allowed to choose actions probabilistically in each node and to transition to a node probabilistically.

Existing approaches for solving infinite horizon problems are exclusively concerned with finding optimal joint controllers of a *fixed size*. The size of a controller is the number of nodes in it. So, if the size of each controller is increased, the expected discounted sum of rewards or the average reward would also potentially increase. Existing approaches include [SC05] which finds an optimal fixed-size deterministic joint controller and [BHZ05] and [ABZ07], which find locally optimal fixed-size stochastic joint controllers. Note that [ABZ07] develop a nonlinear program, similar in structure to **NLP1** presented in Chapter 3, whose solution is a (locally) optimal stochastic joint controller of a fixed size.

The key feature of our approach for the finite horizon case was the use of a finite horizon policy in the sequence form instead of in the canonical tree form. In order to adapt our approach to the infinite horizon case, we are required to conceive a controller in the sequence-form, and use that instead of the canonical form described above. Once the characterization of the sequence-form of a controller is obtained, we believe that we can conceive mathematical programs (0-1 MILPs, in particular) for finding an optimal joint controller in the sequence-form. Additionally, we believe that such a joint controller is not required to be of a fixed size, but can assume the size required in order to achieve the above criteria.

Part III

Appendices / Annexes

Appendix A

An Algorithm For POSGs

A.1 Introduction

The problem of finding a Nash Equilibrium in a 2-agent Dec-POMDP or a 2-agent partially observable stochastic game (POSG) corresponds to a mathematical program called a *linear complementarity problem* (LCP) [Mur88]. In this annex, we shall describe this LCP. The LCP is derived from the necessary conditions for a joint policy to be a Nash Equilibrium. We shall then adapt the 0-1 MILP presented in Chapter 4 (5.68)-(5.79) for finding a Nash Equilibrium of a 2-agent DEC-POMDP/POSG.

A POSG is the competitive analogue of a DEC-POMDP. It is different from a DEC-POMDP only in the reward function. In a POSG each agent has his own reward function while in a DEC-POMDP they have a common reward function. So whereas a DEC-POMDP is defined as the tuple (I, S, $\{A_i\}$, $\{O_i\}$, $\mathbb{P}$, $\mathbb{G}$, R), POSG is defined as the tuple (I, S, $\{A_i\}$, $\{O_i\}$, $\mathbb{P}$, $\mathbb{G}$, $\{R_i\}$).

Since the reward functions are different for the agents, the value of a joint history in a POSG is also different for the agents. We shall denote the value of a joint history j for agent i by $\mathcal{R}_i(\alpha, j)$. It is of course computed analogously to the value of a joint history in a DEC-POMDP. The only change to be made to this equation is that the common reward function R must be replaced by R_i when computing the value of agent i. The definition of a conditional probability Ψ of a joint history remains the same be it in a DEC-POMDP or a POSG.

Due to the different reward functions in a POSG, a joint policy may give different values to different agents. The value of a joint policy p for agent i shall be denoted by $\mathcal{V}_i(\alpha, p)$. A **Nash Equilibrium** of a POSG is a joint policy $p \in X$ such that,

$$\mathcal{V}_i(\alpha, p) - \mathcal{V}_i(\alpha, (p'_i, p_{-i}) \quad \geq \quad 0, \quad \forall i \in I \; \forall p'_i \in X_i \tag{A.1}$$

That is,

$$\sum_{h \in \mathcal{E}_i} \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}_i(\alpha, (h, j')) \prod_{k \in I \setminus \{i\}} p_k(j'_k) \big\{ p_i(h) - p'_i(h) \big\} \quad \geq \quad 0, \quad \forall i \in I, \forall p'_i \in X_i \tag{A.2}$$

Thus, if a joint policy is a Nash equilibrium, no agent can increase his own expected reward by changing his own policy in the joint policy given that other agents do not change their respective policies in the joint policy.

A.2 A Linear Complementarity Problem

An LCP is defined as a pair (q, M) where q is an m-vector and M is a $m \times m$ matrix. Solving the LCP implies that we are required to find m-vectors w and z such that,

$$\begin{aligned} w - Mz &= q && \text{(A.3)} \\ w^T z &= 0 && \text{(A.4)} \\ w, z &\geq 0 && \text{(A.5)} \end{aligned}$$

Since w and z are nonnegative, $w^T z = 0$ actually means $w_i z_i = 0$, for $i = 1, 2, \ldots, m$.

An alternative representation of an LCP does away with the vector w. The vector w can be thought of as a vector of slack variables representing the difference of the quantities Mz and $-q$. That is, letting $w = Mz + q$, we obtain another form of the LCP whose solution now only requires the m-vector z. That is, solving an LCP implies we are required to find an m-vector z such that,

$$\begin{aligned} Mz + q &\geq 0 && \text{(A.6)} \\ z^T(Mz + q) &= 0 && \text{(A.7)} \\ z &\geq 0 && \text{(A.8)} \end{aligned}$$

A solution to an LCP can be obtained through obtained through Lemke's complementary pivoting algorithm [Lem65]. Our objective shall be convert the necessary conditions for a 2-agent joint policy in a DEC-POMDP or a POSG to the standard LCP form (A.3)-(A.5).

The derivation of the LCP is the same for a 2-DEC-POMDP and a 2-agent POSG. We shall therefore adopt the following common notation. For each agent $i \in \{1, 2\}$, we define an $n_1 \times n_2$ matrix $\mathcal{D}_i$ whose rows are labeled by histories of agent 1, whose columns are labeled by histories of agent 2 and whose entries are as follows. For each $h_1 \in H_1$ and for each $h_2 \in H_2$,

$$\mathcal{D}_i(h_1, h_2) = \begin{cases} \mathcal{R}_i(\alpha, (h_1, h_2)), & \text{if } h_1 \text{ and } h_2 \text{ are both terminal} \\ 0, & \text{otherwise} \end{cases} \quad \text{(A.9)}$$

Note that the number of histories of agent i is denoted by n_i. In the case of a DEC-POMDP, $\mathcal{D}_1 = \mathcal{D}_2$. Secondly, in a DEC-POMDP, $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}$. Henceforth in this section, the discussion shall apply to a DEC-POMDP as well as a POSG.

Following Chapter 5, an n_1-vector x_1 and an n_2-vector x_2 together constitute a Nash Equilibrium if there exist an n_1-vector w_1 and an n_2-vector w_2, and an m_1-vector y_1 and an m_2-vector

y_2 such that, for $i = 1, 2$,

$$\sum_{a \in A_i} x_i(a) = 1 \tag{A.10}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{A.11}$$

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) = w_i(h), \quad \forall h \in \mathcal{N}_i \tag{A.12}$$

$$y_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}_i(\alpha, (h, j'))x_{-i}(j') = w_i(h), \quad \forall h \in \mathcal{E}_i \tag{A.13}$$

$$x_i(h)w_i(h) = 0, \quad \forall h \in \mathcal{H}_i \tag{A.14}$$

$$x_i(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{A.15}$$

$$w_i(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{A.16}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad \forall \iota \in \mathcal{I}_i \tag{A.17}$$

In matrix notation, these become,

$$C_1 x_1 = c_1 \tag{A.18}$$

$$C_2 x_2 = c_2 \tag{A.19}$$

$$C_1' y_1 - \mathcal{D}_1 x_2 = w_1 \tag{A.20}$$

$$C_2' y_2 - \mathcal{D}_2' x_1 = w_2 \tag{A.21}$$

$$x_1' w_1 = 0 \tag{A.22}$$

$$x_2' w_2 = 0 \tag{A.23}$$

$$x_1, w_1 \geq 0 \tag{A.24}$$

$$x_2, w_2 \geq 0 \tag{A.25}$$

$$y_1, y_2 \quad \text{free} \tag{A.26}$$

In order to bring this system to the standard LCP form (A.3)-(A.5), we must replace the two vectors y_1 and y_2 by nonnegative vectors. In an LCP, only nonnegative variables are allowed, and these two vectors are not constrained to be nonnegative in (A.18)-(A.26). So, we express each vector y_i as the difference of two nonnegative vectors y_{i1} and y_{i2}. That is, we let,

$$y_1 = y_{11} - y_{12} \tag{A.27}$$

$$y_2 = y_{21} - y_{22} \tag{A.28}$$

$$y_{11}, y_{12} \geq 0 \tag{A.29}$$

$$y_{21}, y_{22} \geq 0 \tag{A.30}$$

With these additions, the system (A.18)-(A.26) is bought to the standard LCP form (A.3)-(A.5) in which w, M, z and q are as follows.

$$w = \begin{pmatrix} w_1 \\ w_2 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} M = \begin{pmatrix} 0 & -\mathcal{D}_1 & C_1' & -C_1' & 0 & 0 \\ -\mathcal{D}_2' & 0 & 0 & 0 & C_2' & -C_2' \\ -C_1 & 0 & 0 & 0 & 0 & 0 \\ C_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -C_2 & 0 & 0 & 0 & 0 \\ 0 & C_2 & 0 & 0 & 0 & 0 \end{pmatrix} z = \begin{pmatrix} x_1 \\ x_2 \\ y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \end{pmatrix} q = \begin{pmatrix} 0 \\ 0 \\ c_1 \\ -c_1 \\ c_2 \\ -c_2 \end{pmatrix} \tag{A.31}$$

A solution to this LCP yields a sample Nash Equilibrium of the 2-agent DEC-POMDP/POSG. Note that for each agent $i = 1, 2$ the vectors x_i, y_{i1}, y_{i2}, and w_i are nonnegative. A solution (x_1^*, x_2^*) of the program is a T-period joint policy that is a Nash Equilibrium, whose value for the first agent is $x_1^* \mathcal{D}_1 x_2^*$ and for the second agent is $x_1^* \mathcal{D}_2 x_2^*$. We can solve this LCP using Lemke's Algorithm [Lem65]. In conceiving this LCP, we are in fact following the KMvS approach [KMvS94]. An LCP that finds a Nash Equilibrium of an extensive game is analogous (A.31). An alternative approach to KMvS for finding a Nash Equilibrium of a 2-agent DEC-POMDP/POSG or extensive game is as follows.

A.3 An 0-1 Mixed Integer Linear Program

In Chapter 5, we presented the 0-1 MILP **MILP3** for finding an optimal 2-agent joint policy. The constraints of this program are just the necessary conditions for an n_1-vector x_1 and an n_2-vector x_2 to constitute a Nash Equilibrium. Due to the objective function of the program, not only does the program find a Nash Equilibrium (x_1, x_2) but moreover, the Nash Equilibrium found is also an optimal joint policy. So that if we remove the objective function from the program, a solution of the program would be only a Nash Equilibrium.

In the case of a 2-agent POSG, therefore, a Nash Equilibrium can be found by removing the objective function from (5.68)-(5.79) and replacing the common joint history value function $\mathcal{R}$ by, for each agent i, $\mathcal{R}_i$. The changed program is as follows: for each $i = 1, 2$:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{A.32}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(hoa) = 0, \quad i = 1, 2, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{A.33}$$

$$y_i(\iota(h)) - \sum_{o \in O_i} y_i(ho) = w_i(h), \quad i = 1, 2, \forall h \in \mathcal{N}_i \tag{A.34}$$

$$y_1(\iota(h)) - \sum_{h \in \mathcal{E}_2} \mathcal{R}_1(\alpha, (h, h')) x_2(h') = w_1(h), \quad \forall h \in \mathcal{E}_1 \tag{A.35}$$

$$y_2(\iota(h)) - \sum_{h \in \mathcal{E}_1} \mathcal{R}_2(\alpha, (h', h)) x_1(h') = w_2(h), \quad \forall h \in \mathcal{E}_2 \tag{A.36}$$

$$x_i(h) \leq 1 - b_i(h), \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{A.37}$$

$$w_i(h) \leq \mathcal{U}_i(h) b_i(h), \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{A.38}$$

$$x_i(h) \geq 0, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{A.39}$$

$$w_i(h) \geq 0, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{A.40}$$

$$b_i(h) \in \{0, 1\}, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{A.41}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad i = 1, 2, \forall \iota \in \mathcal{I}_i \tag{A.42}$$

The 0-1 MILP (A.32)-(A.42) shall be henceforth referred to as **MILP6**. Given a solution (x^*, y^*, w^*, b^*) to **MILP6**, (x_1^*, x_2^*) is a T-period joint policy that is a Nash Equilibrium of the 2-agent DEC-POMDP/POSG, whose value for the first agent is

$$\mathcal{V}_1(\alpha, (x_1^*, x_2^*)) = \sum_{j \in \mathcal{E}} \mathcal{R}_1(\alpha, j) x_1^*(j_1) x_2^*(j_2) \tag{A.43}$$

$$= y_1^*(\varnothing) \tag{A.44}$$

and whose value for the second agent is,

$$\mathcal{V}_2(\alpha, (x_1^*, x_2^*)) = \sum_{j \in \mathcal{E}} \mathcal{R}_2(\alpha, j) x_1^*(j_1) x_2^*(j_2) \quad (A.45)$$

$$= y_2^*(\varnothing) \quad (A.46)$$

Upper bounds $\mathcal{U}_i$s on the regrets of histories of the agents are defined identically to those for regrets of histories in a DEC-POMDP.

We can find a socially maximizing Nash Equilibrium (one that maximizes the sum of the values of the joint policies found) by adding the following as the objective function,

$$y_1^T c_1 + y_2^T c_2 = y_1(\varnothing) + y_2(\varnothing) \quad (A.47)$$

to the program, since as shown in Chapter 5, the duality theorem of linear programming implies that the value of a joint policy for an agent equals the value of the null information set of the agent.

Note that a 0-1 MILP analogous to **MILP6** can be used for solving 2-player extensive games.

A.3.1 The 3-Or-More Agents Case

We can find a Nash Equilibrium of a 3-or-more agents POSG by modifying either of the two 0-1 MILPs **MILP4** and **MILP5**. The changes to be made are as follows. In either program, the objective function must be dropped and each occurrence of $\mathcal{R}$ must be replaced by $\mathcal{R}_i$. The resultant 0-1 MILP would be able to find a Nash Equilibrium of the POSG provided that there exists a *pure* Nash Equilibrium. A pure Nash Equilibrium is a pure joint policy that is a Nash Equilibrium.

Appendix B

Algorithm To Find Correlated Equilibrium

B.1 Introduction

A Nash Equilibrium is a special case of a Correlated Equilibrium [Aum74]. While a Nash Equilibrium is an n-tuple of distributions, each distribution in the tuple being a distribution over an agent's set of policies, a Correlated Equilibrium is a distribution over the set of joint policies. In order to implement a Correlated Equilibrium, the agents require the aid of a trusted intermediary. This intermediary tells each agent privately what policy to use. A Correlated Equilibrium is such that the agents have no incentive to disobey the intermediary. An agent cannot profit by disobeying the intermediary. In implementing a Nash Equilibrium, an intermediary is superfluous because the instruction of the intermediary to an agent cannot be different from the precepts of the distribution of the agent in the Nash Equilibrium.

A Correlated Equilibrium as a solution of an infinite horizon DEC-POMDP has been investigated by [Ber05]. In this annex, we shall describe a linear program that finds a Correlated Equilibrium of an n-agent finite horizon DEC-POMDP, $n \geq 2$. The program can also be used for finding a Correlated Equilibrium of an n-agent finite horizon POSG. The program finds a distribution over the set of joint histories. The rest of the annex is organized as follows. We first define a Correlated Equilibrium in a normal form game and a linear program of finding it. We then define a Correlated Equilibrium in a finite horizon DEC-POMDP and the linear program for finding it.

B.2 Correlated Equilibrium: Normal Form Game

A normal form game is defined as the tuple $(I, \{A_i\}, \{R_i\})$ where I is set of n agents, A_i is the set of actions of agent $i \in I$ and R_i is agent i's reward function defined from A to $\mathbb{R}$ where, A is the set of joint actions, $\times_{i \in I} A_i$. The set of i-reduced joint actions is denoted by A_{-i}.

A **Correlated Equilibrium** of the normal form game is a probability distribution p over the set of joint actions A that satisfies the following conditions: For each agent $i \in I$, for each pair

of actions $a, a' \in A_i$,

$$\sum_{b \in A_{-i}} p(a,b)(R_i(a,b) - R_i(a',b)) \quad \geq \quad 0 \tag{B.1}$$

where $p(a,b)$ is the probability of the joint action (a, b) in p.

Note that while every Nash Equilibrium is necessarily (by definition) a Correlated Equilibrium, not every Correlated Equilibrium is a Nash Equilibrium. There are three methods of finding a Correlated Equilibrium of a normal form game. First, by solving a linear program that directly implements the definition of a Correlated Equilibrium (as given below). Second, through a reinforcement learning procedure [HMC00]. Third, through an ellipsoid-based algorithm [Pap05].

The following objective-less linear program implements the definition of a Correlated Equilibrium and thereby finds a Correlated Equilibrium for a given normal form game.

$$\sum_{b \in A_{-i}} x(a,b)(R_i(a,b) - R_i(a',b)) \quad \geq \quad 0, \quad \forall i \in I, \forall a, a' \in A_i \tag{B.2}$$

$$\sum_{\tilde{a} \in A} x(\tilde{a}) \quad = \quad 1 \tag{B.3}$$

$$x(\tilde{a}) \quad \geq \quad 0, \quad \forall \tilde{a} \in A \tag{B.4}$$

A solution x^* to this LP is a Correlated Equilibrium of the game. The expected reward to agent i in this Correlated Equilibrium is $\sum_{\tilde{a} \in A} x^*(\tilde{a}) R_i(\tilde{a})$.

In order to implement the Correlated Equilibrium, a trusted intermediary uses the following scheme. Let the number of agents be 2. Let each agent have two actions a and b . Let the 4 joint actions be aa, ab, ba and bb. Suppose a Correlated Equilibrium of the game is (0.2, 0.4, 0.15, 0.25) for the probabilities of the 4 joint actions. The intermediary will sample this probability distribution and draw out a joint action. Say it is ba. He will tell agent 1 privately only that agent 1 should play action b. Similarly, he will tell agent 2 privately only that agent 2 should play action a. Neither agent knows what has been told to the other agent.

The intermediary is required to be a trusted one for two reasons. First, he should sample the probability distribution honestly. Second, he should report the sampled joint action honestly to all agents. If the agents are satisfied that the intermediary is honest in these two aspects, then no agent has an incentive to disobey the intermediary. It is vital that the intermediate reveal to an agent which action he should take privately. If an agent finds out the action suggested to the other agents, he can use that information to switch to another action that gives him personally a higher reward. The following example, taken from [Pap05], illustrates this point. Consider the following 2-agent, 2-action game called the game of *chicken*. Each agent has 2 actions stop and go. The rewards of the two agents for joint actions are given as follows.

	stop	go
stop	4, 4	1, 5
go	5, 1	0, 0

So, if agent 1 chooses go and agent 2 chooses stop, agent 1 gets a reward of 5 while agent 2 gets a reward of 1.

The game has two pure Nash equilibria: the first one in which the agents choose the joint action (stop, go) and the second in which the agents choose the joint action (go, stop). The second agent prefers the first equilibrium to the second since his reward is 5 in the first and 1 in the second. On the other hand, the first agent prefers the second equilibrium to the first because his reward is 5 in the second and 1 in the first. If both agents play according to their preferred equilibrium, they will end up playing (go, go) resulting in rewards of 0 to both agents. A compromise is the following Correlated Equilibrium

$$\begin{pmatrix} 1/3 & 1/3 \\ 1/3 & 0 \end{pmatrix} \tag{B.5}$$

where the intermediary chooses the three joint actions (stop, stop), (go, stop) and (stop, go) with equal probability.

Now it is important that neither agent know which joint action has been chosen by the intermediary. Suppose the intermediary picks (stop, stop). He is meant to tell agent 1 privately to choose stop and to tell agent 2 privately to choose stop. Suppose agent 1 comes to know that the intermediary has chosen (stop, stop). Then, he will clearly disobey the intermediary and switch to go to obtain a higher reward (5 rather than 4) knowing that agent 2 will choose stop. On the other hand, if agent 1 did not know which action was suggested to agent 2 (it could be stop or go), he will not have any incentive to switch because by switching he risks getting a lower reward (0 rather than 4). Hence, the intermediary is required to privately tell each agent which action to take.

B.3 Correlated Equilibrium: DEC-POMDP

A Correlated Equilibrium in a normal form game is defined as a probability distribution over A, the set of joint actions. A Correlated Equilibrium of a DEC-POMDP can be defined as a probability distribution over Π^T, the set of (pure) T-period canonical joint policies. Formally, a probability distribution δ over Π^T is a Correlated Equilibrium at α of the DEC-POMDP if it satisfies the following conditions:

$$\sum_{\sigma \in \Pi^T_{-i}} \delta(\pi, \sigma)(V(\alpha, (\pi, \sigma)) - V(\alpha, (\pi', \sigma))) \quad \geq \quad 0, \quad \forall i \in I, \forall \pi, \pi' \in \Pi^T_i \tag{B.6}$$

Note that for a POSG, a Correlated Equilibrium is analogously defined; we only need to replace V by V_i.

Similarly, we can also define a Correlated Equilibrium of a DEC-POMDP in terms of joint policies in the sequence-form. But here the definition is slightly different. A Correlated Equilibrium is defined as a weight distribution over H, the set of joint histories. This weight distribution should be such that it corresponds to the weight distribution achieved by some POMDP policy (in the sequence-form) of the DEC-POMDP. A POMDP policy of the DEC-POMDP is policy of the POMDP obtained from the DEC-POMDP. We have defined a T-period POMDP policy of the DEC-POMDP in Chapter 6. We do so again here for convenience. A **T-period POMDP**

policy in the sequence-form to be a function q from $\mathcal{H}$ to [0, 1] such that,

$$\sum_{a \in A} q(a) = 1 \tag{B.7}$$

$$-q(j) + \sum_{a \in A} q(joa) = 0, \quad \forall j \in \mathcal{N}, \forall o \in O \tag{B.8}$$

where joa denotes the joint history obtained on concatenating the joint observation o and the joint action a to joint history j. Note that this definition is given in terms of joint actions and joint observations, but it applies analogously to a given POMDP.

A T-period POMDP policy θ of the DEC-POMDP is a Correlated Equilibrium at α of the DEC-POMDP if it satisfies the following conditions:

$$\sum_{j' \in \mathcal{E}_{-i}} \theta(h, j')(\mathcal{R}(\alpha, (h, j')) - \mathcal{R}(\alpha, (h', j'))) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{E}_i, \forall h' \in C(h) \tag{B.9}$$

where $C(h)$ is the set of cohistories of h (that is, those histories that differ from h only in their last action). Note that for a POSG, a Correlated Equilibrium is analogously defined; we only need to replace $\mathcal{R}$ by $\mathcal{R}_i$.

It is easy to see that the set of conditions (B.6) are entirely equivalent to the set of conditions (B.9). In other words, the two are interchangeable. Each implies the other. Thereby, the following objective-less linear program finds a Correlated Equilibrium of a DEC-POMDP,

$$\sum_{a \in A} z(a) = 1 \tag{B.10}$$

$$-z(j) + \sum_{a \in A} z(joa) = 0, \quad \forall j \in \mathcal{N}, \forall o \in O \tag{B.11}$$

$$\sum_{j' \in \mathcal{E}_{-i}} z(h, j')(\mathcal{R}(\alpha, (h, j')) - \mathcal{R}(\alpha, (h', j'))) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{E}_i, \forall h' \in C(h) \tag{B.12}$$

$$z(j) \geq 0, \quad \forall j \in \mathcal{H} \tag{B.13}$$

The program contains one variable $z(j)$ for every joint history j of length less than or equal to T. A solution z^* of the program is a Correlated Equilibrium of the DEC-POMDP.

The one major advantage of a Correlated Equilibrium is that it can be computed through a mere linear program. We do not require a (mixed) integer linear program to find it. Hence, the complexity of finding a Correlated Equilibrium is polynomial and not in the class NP. Note that we can also find a Correlated Equilibrium with the maximum value by adding the following objective function to the LP,

$$\text{maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{B.14}$$

B.3.1 Implementing A Correlated Equilibrium

As in a normal-form game, implementing a Correlated Equilibrium requires an intermediary. During the control of the Markov process, the intermediary is required to suggest to the agents which actions to take in each step based on the computed Correlated Equilibrium and the histories of the agents uptill that period. However, the intermediary does not reveal to any agent the actions taken by and the observations received by the other agents at any step. Hence, this aspect of the decentralization is retained. The intermediary can be thought of as a centralized coordinator.

The following example shows how a Correlated Equilibrium for a 2-agent DEC-POMDP for horizon 2 is implemented. We let the set of actions of the first agent be $\{a, b\}$ and that of the second agent be $\{c, d\}$. The set of observations of the two agents is the same, $\{u, v\}$. Suppose a Correlated Equilibrium is found according to the following table, which lists the weights of joint histories of lengths less than or equal to 2 in the equilibrium.

joint history	weight	joint history	weight
ac	0.4	*bd*	0.6
acuuac	0.2	*acuubc*	0.2
acuvbd	0.4	*acvubc*	0.4
acvvac	0.1	*acvvbc*	0.2
acvvbd	0.1	*bduuad*	0.6
bduvac	0.6	*bdvubc*	0.6
bdvvad	0.6		

At the first step, the intermediary will sample joint histories of length 1 (i.e., joint actions). He will sample *ac* and *bd* with probabilities 0.4 and 0.6 respectively. Suppose his sample is *ac*. He will tell agent 1 privately to take action *a* and he will tell agent 2 privately to take *c*. So, at the first step, the agents will execute the joint action *ac*. At the second step, suppose agent 1 receives observation v and suppose the second agent also receives observation v. The intermediary's sample space is now composed of the three joint histories *acvvac*, *acvvbc* and *acvvbd*. The normalized weights of these joint histories in the Correlated Equilibrium are respectively, 0.1/0.4, 0.2/0.4 and 0.1/0.4. The intermediary will sample this space according to these normalized weights. Suppose his sample is *acvvbc*. Then, he will tell agent 1 privately to take action *b* and he will tell agent 2 privately to take action *c*. Hence, as required in a DEC-POMDP, the agents do not exchange any information during the control. Each agent neither knows the actions taken by the other agents nor the observations received by them.

Appendix C

Nash Equilibrium Conditions

C.1 The Kuhn-Tucker Theorem

In Chapter 5, we obtained the necessary conditions for a joint policy to be a Nash equilibrium using the theorem of linear programming duality. These conditions can also be obtained through the *Kuhn-Tucker (KT) Theorem* [Dor61], [Lue84]. This theorem defines the necessary conditions for a point to be a locally optimal solution to a nonlinear program (NLP).

Consider the following NLP:

$$\text{maximize} \quad f(x) \tag{C.1}$$

subject to,

$$\begin{aligned} Ax &= b & \text{(C.2)} \\ x &\geq 0 & \text{(C.3)} \end{aligned}$$

Here f is a function from $\mathbb{R}^{n'}$ to $\mathbb{R}$. A is an $m' \times n'$ matrix and b is an m'-vector.

The *Lagrangian* of this NLP is defined as the function,

$$\Upsilon(x, \lambda, \mu) = f(x) - \lambda'(Ax - b) + \mu' x \tag{C.4}$$

where λ is an m'-vector and μ is an n'-vector. The two vectors are called vectors of *Lagrange multipliers*. Note that in λ' and μ' denote the transpose of λ and μ respectively.

According to the KT Theorem, $x^* \in \mathbb{R}^{n'}$ is a local maximum point of the NLP, if there holds,

$$\begin{aligned} \frac{\partial \Upsilon}{\partial x}(x^*) &= 0 & \text{(C.5)} \\ \mu' x^* &= 0 & \text{(C.6)} \\ Ax^* &= b & \text{(C.7)} \\ x &\geq 0 & \text{(C.8)} \\ \mu &\geq 0 & \text{(C.9)} \end{aligned}$$

The partial derivative of Υ with respect to x is,

$$\frac{\partial \Upsilon}{\partial x} = \frac{\partial f}{\partial x} - A'\lambda + \mu \tag{C.10}$$

Therefore, according to the KT Theorem, $x^* \in \mathbb{R}^{n'}$ is a local maximum point of the NLP, if there holds,

$$\frac{\partial f(x^*)}{\partial x} - A^T\lambda + \mu = 0 \tag{C.11}$$

$$\mu' x^* = 0 \tag{C.12}$$

$$Ax^* = b \tag{C.13}$$

$$x^* \geq 0 \tag{C.14}$$

$$\mu \geq 0 \tag{C.15}$$

These conditions are also called the **KT conditions**.

C.2 Applying the KT Theorem to NLP1

We shall now apply the KT Theorem to the nonlinear program **NLP1** presented in Chapter 3 whose globally optimal solution is an optimal T-period joint policy.

For each agent $i \in I$, let C_i denote an $m_i \times n_i$ matrix whose entries are the coefficients of the policy constraints of agent i (Chapter 2). Let c_i denote an m_i vector whose first entry is 1 and remaining entries are all 0s. Then, the policy constraints of agent i can be succinctly represented as,

$$C_i x_i = c_i \tag{C.16}$$

$$x_i \geq \mathbf{0} \tag{C.17}$$

where x_i is an n_i-vector and $\mathbf{0}$ is an n_i-vector each of whose entries is a 0. Then, **NLP1** (Chapter 3) can be rewritten as,

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{C.18}$$

Subject to,

$$C_i x_i = c_i, \quad \forall i \in I \tag{C.19}$$

$$x_i \geq \mathbf{0}, \quad \forall i \in I \tag{C.20}$$

Let,

$$f(x) = \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{C.21}$$

The Lagrangian of this NLP is the function,

$$\Upsilon(x_1, x_2, \ldots, x_n, \lambda_1, \lambda_2, \ldots, \lambda_n, \mu_1, \mu_2, \ldots, \mu_n) = f(x) - \sum_{i \in I} \lambda_i'(C_i x_i - c_i) + \sum_{i \in I} \mu_i' x_i$$

For each $i \in I$, the first-order partial derivative of this function with respect to x_i is,

$$\frac{\partial \Upsilon}{\partial x_i} = \frac{\partial f}{\partial x_i} - C_i' \lambda_i + \mu_i \tag{C.22}$$

where, if h is a terminal history,

$$\frac{\partial f}{\partial x_i(h)} = \frac{\partial \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i)}{\partial x_i(h)} \tag{C.23}$$

$$= \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, j) \prod_{k \in I \setminus \{i\}} x_k(j_k') \tag{C.24}$$

and if h is a nonterminal history,

$$\frac{\partial f}{\partial x_i(h)} = 0 \tag{C.25}$$

Hence, the first-order partial derivative of this function with respect to x_i,

$$\frac{\partial f}{\partial x_i} - C_i' \lambda_i + \mu_i \tag{C.26}$$

is, if h is a terminal history, becomes,

$$\sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, j) \prod_{k \in I \setminus \{i\}} x_k(j_k') - \lambda_i(\iota(h)) + \mu_i(h) \tag{C.27}$$

and is, if h is a nonterminal history, becomes,

$$\big\{\lambda_i(\iota(h)) - \sum_{o \in O_i} \lambda_i(ho)\big\} + \mu_i(h) \tag{C.28}$$

According to the KT Theorem, the first-order partial derivative of Υ vanishes. Therefore, for each agent $i \in I$, for each history $h \in H_i$ there holds,

$$-\big\{\lambda_i(\iota(h)) - \sum_{o \in O_i} \lambda_i(ho)\big\} + \mu_i(h) = 0, \quad \text{if } h \in \mathcal{N}_i \tag{C.29}$$

$$\sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, j) \prod_{k \in I \setminus \{i\}} x_k(j_k') - \lambda_i(\iota(h)) + \mu_i(h) = 0, \quad \text{if } h \in \mathcal{E}_i \tag{C.30}$$

That is,

$$\mu_i(h) = \big\{\lambda_i(\iota(h)) - \sum_{o \in O_i} \lambda_i(ho)\big\}, \quad \text{if } h \in \mathcal{N}_i \tag{C.31}$$

$$\mu_i(h) = \lambda_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, j) \prod_{k \in I \setminus \{i\}} x_k(j_k'), \quad \text{if } h \in \mathcal{E}_i \tag{C.32}$$

Therefore, an n-tuple of vectors x_1, x_2, …, x_n where each x_i is an n_i vector constitutes a Nash Equilibrium if there exist an n-tuple of vectors λ_1, λ_2, …, λ_n where each λ_i is an m_i-vector and

an n-tuple of vectors μ_1, μ_2, ..., μ_n where each μ_i is an n_i-vector such that, for each agent $i \in I$ there holds,

$$\mu_i(h) = \lambda_i(\iota(h)) - \sum_{o \in O_i} \lambda_i(ho), \quad \forall h \in \mathcal{N}_i \tag{C.33}$$

$$\mu_i(h) = \lambda_i(\iota(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, j) \prod_{k \in I \setminus \{i\}} x_k(j'_k), \quad \forall h \in \mathcal{E}_i \tag{C.34}$$

$$C_i x_i = c_i \tag{C.35}$$

$$\mu_i(h) x_i(h) = 0, \forall h \in \mathcal{H}_i \tag{C.36}$$

$$x_i(h) \geq 0, \forall h \in \mathcal{H}_i \tag{C.37}$$

$$\mu_i(h) \geq 0, \forall h \in \mathcal{H}_i \tag{C.38}$$

$$y_i(\iota) \in [-\infty, +\infty], \quad \forall \iota \in \mathcal{I}_i \tag{C.39}$$

These conditions are of course the identical to (5.48)-(5.55) derived in Chapter 5.

Appendix D

Notations

D.1 DEC-POMDP Notation

I	The set of agents
S	The set of states
$\Delta(S)$	The set of probability distributions over S
A_i	The set of actions of agent i
A	The set of joint actions
O_i	The set of observations of agent i
O	The set of joint observations
$\mathbb{P}$	The state transition function
$\mathbb{G}$	The joint observation function
R	The reward function
α	The initial state

D.2 Canonical Form Notation

$\overline{O}_i^t$	The set of sequences of t observations of agent i
$\overline{O}^t$	The set of sequences of t joint observations
Π_i^t	The set of t-period policies of agent i
Π_{-i}^t	The set of i-reduced t-period joint policies
Π^t	The set of t-period joint policies
$V(\alpha, \sigma)$	The value of joint policy σ for initial state $\alpha \in \Delta(S)$

D.3 Sequence Form Notation

$\mathcal{H}_i^t$	The set of histories of length t of agent i
$\mathcal{H}^t$	The set of histories of lengths less than or equal to T of agent i
$\mathcal{E}_i$	The set of histories of length T (i.e. terminal histories) of agent i
$\mathcal{N}_i$	The set of histories of lengths less than T (i.e. non-terminal histories) of agent i
$\mathcal{H}_{-i}^t$	The set of i-reduced joint histories of length t
$\mathcal{E}_{-i}$	The set of i-reduced terminal joint histories
$\mathcal{H}^t$	The set of joint histories of length t
$\mathcal{H}$	The set of joint histories of lengths less than or equal to T
$\mathcal{E}$	The set of joint histories of length T (i.e., terminal joint histories)
$\mathcal{N}$	The set of joint histories of lengths less than T (i.e., non-terminal joint histories)
X_i	The set of T-period policies of agent i
X_{-i}	The set of T-period i-reduced joint policies
$\mathcal{R}(\alpha, j)$	The value of joint history j for initial state $\alpha \in \Delta(S)$
$\Psi(\alpha, j)$	The joint observations sequence probability of j for initial state $\alpha \in \Delta(S)$
$\mathcal{V}(\alpha, p)$	The value of joint policy p for initial state $\alpha \in \Delta(S)$
$p_i(h)$	The weight of history h in policy p_i of agent i
$p(j)$	The weight of joint history j in joint policy p
$p_{-i}(j')$	The weight of i-reduced joint history j' in i-redcued joint policy p_{-i}
$\mathcal{I}_i^t$	The set of information sets of length t of agent i
$\mathcal{I}_i$	The set of information sets of lengths less than T of agent i
$\lambda_i^*(\iota, q_{-i})$	The value of information set ι of agent i given i-reduced joint policy q_{-i}
$\mu_i(h, q_{-i})$	The regret of history h of agent i given i-reduced joint policy q_{-i}
$\mathcal{U}_i(h)$	The upper bound on the regret of history h of agent i
$C(h)$	The set of co-histories of history h

Bibliography

[ABZ07] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing Memory-Bounded Controllers for Decentralized POMDPs. *In Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI-07)*, 2007.

[ADC07] Raghav Aras, Alain Dutech, and François Charpillet. Mixed Integer Linear Programming For Exact Finite-Horizon Planning In Decentralized POMDPs. *In Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 2007.

[AM80] B.D.O. Anderson and J.B. Moore. Time-varying Feedback Laws For Decentralized Control. *Nineteenth IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, 19(1):519 – 524, 1980.

[AMR05] Salman Azhar, Andrew McLennan, and John Reif. Computation of Equilibria in Noncooperative Games. *Computers and Mathematics with Applications*, 50:823 – 854, 2005.

[Aum67] Robert Aumann. Mixed and Behavior Strategies in Infinite Extensive Games. *Advances in Game Theory, Annals of Mathematics Studies (M. Dresher, L. S. Shapley, and A. W. Tucker, editors)*, 52:627–650, 1967.

[Aum74] Robert J. Aumann. Subjectivity And Correlation In Randomized Strategies. *Journal of Mathematical Economics*, 1:67 – 96, 1974.

[Bal65] M.L. Balinski. Integer Programming: Methods, Uses, Computation. *Management Science*, 12(3):253 – 313, 1965.

[BCd08] Abdeslam Boularias and Brahim Chaib-draa. Exact Dynamic Programming For Decentralized POMDPs with Lossless Policy Compression. *In Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 2008.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Ber05] Daniel S. Bernstein. Complexity Analysis And Optimal Algorithms For Decentralized Decision Making. *Ph.D. Thesis, University of Massachusetts Amherst, Amherst, Mass, USA*, 2005.

[BGIZ02] Daniel Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The Complexity Of Decentralized Control Of Markov Decision Processes. *Mathematics of Operations Research*, 27(4):819 – 840, 2002.

[BHZ05] Daniel S. Bernstein, Eric Hansen, and Shlomo Zilberstein. Bounded Policy Iteration for Decentralized POMDPs. *In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 2005.

[BM06] A. Beynier and A.I Mouaddib. An Iterative Algorithm For Solving Constrained Decentralized Markov Decision Processes. *In Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[BMvL96] Jean R Blair, David Mutchler, and Michael van Lent. Perfect Recall and Pruning in Games with Imperfect Information. *Computational Intelligence*, 12:131–154, 1996.

[BSK06] B. Blum, C.R. Shelton, and D. Koller. A Continuation Method for Nash Equilibria in Structured Games. *Journal of Artificial Intelligence Resarch*, 25:457–502, 2006.

[BZLG04] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Solving Transition Independent Decentralized Markov Decision Processes. *Journal of Artificial Intelligence Research*, 22:423 – 455, 2004.

[Cas98a] A.R. Cassandra. A Survey Of POMDP Applications. 1998.

[Cas98b] A.R. Cassandra. Exact And Approximate Algorithms For Partially Observable Markov Decision Processes. *Ph.D. Thesis, Brown University, Providence, RI, USA*, 1998.

[CL06] Randy Cogill and Sanjay Lall. An Approximation Algorithm For The Discrete Team Decision Problem. *SIAM Journal on Control and Optimization,*, 2006.

[CRRL04] Randy Cogill, Michael Rotkowitz, Benjamin Van Roy, and Sanjay Lall. An Approximate Dynamic Programming Approach To Decentralized Control Of Stochastic Systems. *In Proceedings of the Forty-Second Allerton Conference on Communication, Control, and Computing*, 2004.

[CS05] Vincent Conitzer and Tuomas Sandholm. A Generalized Strategy Eliminability Criterion And Computational Methods For Applying It. *In Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 2005.

[CSC02] Iadine Chadès, Bruno Scherrer, and François Charpillet. A Heuristic Approach For Solving Decentralized POMDPs: Assessment On The Pursuit Problem. *In Proceedings of the Sixteenth ACM Symposium on Applied Computing (SAC 2002)*, 2002.

[Dan60] George B. Dantzig. On the Significance of Solving Linear Programming Problems with Some Integer Variables. *Econometrica*, 28(1):30–44, 1960.

[d'E63] F. d'Epenoux. A Probabilistic Production And Inventory Problem. *Management Science*, 10(1):98–108, 1963.

[DK91] John Dickhaut and Todd Kaplan. A Program For Finding Nash Equilibria. *The Mathematica Journal*, pages 87–93, 1991.

[Doo42] J. L. Doob. What Is A Stochastic Process? *The American Mathematical Monthly*, 49(10):648 – 653, 1942.

[Dor61] W. S. Dorn. On Lagrange Multipliers and Inequalities. *Operations Research*, 9(1):95–104, 1961.

[EMGST04] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004)*, pages 136–143, 2004.

[Fis81] Marshall L. Fisher. The Lagrangian Relaxation Method For Solving Integer Programming Problems. *Management Science*, 27(1):1–18, 1981.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, 1987.

[GW01] Srihari Govindan and Robert Wilson. A Global Newton Method To Compute Nash Equilibria. *Journal of Economic Theory*, 110:65–86, 2001.

[Haj84] Bruce Hajek. Optimal Control Of Two Interacting Service Stations. *IEEE Transactions on Automatic Control*, 29:491 – 499, 1984.

[HBZ04] Eric Hansen, Daniel Bernstein, and Shlomo Zilberstein. Dynamic Programming For Partially Observable Stochastic Games. *In Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 709 – 715, 2004.

[HMC00] S. Hart and A. Mas-Colell. A Simple Adaptive Procedure Leading To Correlated Equilibria. *Econometrica*, 68(5):1127–1150, 2000.

[Iba71] Toshihide Ibaraki. Complementary Programming. *Operations Research*, 19(6):1523–1529, 1971.

[Kle80] Victor Klee. Combinatorial Optimization: What Is the State of the Art. *Mathematics of Operations Research*, 5(1):1–26, 1980.

[KM92] Daphne Koller and Nimrod Megiddo. The Complexity of Zero-Sum Games in Extensive Form. *Games and Economic Behavior*, 4:4:528–552, 1992.

[KM96] Daphne Koller and Nimrod Megiddo. Finding Mixed Strategies with Small Supports in Extensive Form Games. *International Journal of Game Theory*, 25(1):73–92, 1996.

[KM03] Daphne Koller and Brian Milch. Multi-Agent Influence Diagrams for Representing and Solving Games. *Games and Economic Behavior*, 45(1):181–221, 2003.

[KMvS94] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast Algorithms for Finding Randomized Strategies in Game Trees. *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)*, pages 750–759, 1994.

[KMvS96] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behaviour*, 14(2), 1996.

[KR87] David Kreps and Garey Ramey. Structural Consistency, Consistency, and Sequential Rationality. *Econometrica*, 55(6):1331–1348, 1987.

[Kuh50] H.W. Kuhn. Extensive Games. *In Proceedings of the National Academy of Sciences*, 36:570–576, 1950.

[KW82] David Kreps and Robert Wilson. Sequential Equilibria. *Econometrica*, 4:863–894, 1982.

[Lan89] Daniel E. Lane. A Partially Observable Model Of Decision Making By Fishermen. *Operations Research*, 37(2):240–254, 1989.

[Law63] Eugene L. Lawler. The Quadratic Assignment Problem. *Management Science*, 9(4):586–599, 1963.

[LD60] A. H. Land and A. G. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):497 – 520, 1960.

[Lem65] C.E. Lemke. Bimatrix Equilibrium Points And Mathematical Programming. *Management Science*, 11(7):681–689, 1965.

[LH64] C.E. Lemke and J.T. Howson. Equilibrium Points Of Bimatrix Games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[Lov91] W.S. Lovejoy. Computationally Feasible Bounds For Partially Observed Markov Decision Processes. *Operations Research*, 39(1):162–175, 1991.

[Lue84] D.G. Luenberger. *Linear and Nonlinear Programming.* Addion-Wesley Publishing Company, Reading, Massachussetts, 1984.

[Mar55] J. Marschak. Elements For A Theory Of Teams. *Management Science*, 1:127 – 137, 1955.

[Meg87] Nimrod Megiddo. On The Complexity Of Linear Programming. *Advances in Economic Theory, Fifth World Congress (Truman F. Bewley, editor)*, 1987.

[Mon82] G.E. Monahan. A Survey of Partially Observable Markov Decision Processes: Theory, Models And Algorithms. *Management Science*, 28(1):1 – 16, 1982.

[Mur88] K.G. Murty. *Linear Complementarity, Linear and Nonlinear Programming.* Heldermann Verlag, Berlin, 1988.

[Nas51] John Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[NTY+03] Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacey Marsella. Taming Decentralized POMDPs: Towards Efficient Policy Computation For Multiagent Settings. *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 705 – 711, 2003.

[NVTY05] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked Distributed POMDPs: A Synthesis Of Distributed Constraint Optimization And POMDPs. *In Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 2005.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* The MIT Press, Cambridge, Mass, 1994.

[OSV08] Frans A. Oliehoek, Matthijs T.J. Spaan, and Nikos Vlassis. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289 – 353, 2008.

[OW96] James M. Ooi and Gregory W. Wornell. Decentralized Control Of A Multiple Access Broadcast Channel: Performance Bounds. *In Proceedings of the IEEE Conference on Decision and Control*, 1:293 – 298, 1996.

[Pap80] Christos H. Papadimitriou. On The Complexity Of Integer Programming. *Journal of the Association for Computing Machinery*, 28(4):765 – 768, 1980.

[Pap05] Christos H. Papadimitriou. Computing Correlated Equilibria In Multi-Player Games. *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC '05)*, 2005.

[PRA01] G.L. Peterson, J.H. Reif, and S. Azhar. Lower Bounds for Multiplayer Noncooperative Games of Incomplete Information. *Computers and Mathematics with Applications*, 41:957 – 992, 2001.

[PRA02] G.L. Peterson, J.H. Reif, and S. Azhar. Decision Algorithms for Multiplayer Non-Cooperative Games of Incomplete Information. *Computers and Mathematics with Applications*, 43:179 – 206, 2002.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1982.

[PT87] Christos H. Papadimitriou and John Tsitsiklis. The Complexity Of Markov Decision Processes. *Mathematics of Operations Research*, 12 (3):441 – 450, 1987.

[PT02] David Pynadath and Milind Tambe. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories And Models. *Journal of Artificial Intelligence Research*, 2002.

[Put94] Martin L. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[PZ07a] Marek Petrik and Shlomo Zilberstein. Anytime Coordination Using Separable Bilinear Programs. *In Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI 2007)*, 2007.

[PZ07b] Marek Petrik and Shlomo Zilberstein. Average-Reward Decentralized Markov Decision Processes. *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[Rad59] Roy Radner. The Application Of Linear Programming To Team Decision Problems. *Management Science*, 5:143–150, 1959.

[Rei84] John Reif. The Complexity of Two Player Games of Incomplete Information. *Journal of Computer and System Sciences*, 29(2):274 – 310, 1984.

[Rom62] I. V. Romanovskii. Reduction Of A Game With Complete Memory To A Matrix Game. *Soviet Mathematics*, 3:678 – 681, 1962.

[Ros83] Zvi Rosberg. Optimal Decentralized Control In A Multiaccess Channel With Partial Information. *IEEE Transactions on Automatic Control*, 28:187 – 193, 1983.

[Ros86] J. Ben Rosen. Solution Of A General LCP By 0-1 Mixed Integer Programming. *Computer Science Tech Report, University of Minnesota*, 1986.

[SB98] Richard Sutton and Andy Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Mass., 1998.

[SC05] Daniel Szer and François Charpillet. An Optimal Best-First Search Algorithm For Solving Infinite Horizon DEC-POMDPs. *In Proceedings of the Sixteenth European Conference on Machine Learning (ECML 2005)*, 2005.

[SC06] Daniel Szer and François Charpillet. Point-based Dynamic Programming for DEC-POMDPs. *In Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[SCZ05] Daniel Szer, François Charpillet, and Shlomo Zilberstein. MAA*: A Heuristic Search Algorithm For Solving Decentralized POMDPs. *In Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, pages 576 – 583, 2005.

[SGC05] Tuomas Sandolm, Andrew Gilpin, and Vincent Conitzer. Mixed Integer Programming Methods For Finding Nash Equilibria. *In Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 2005.

[Sha74] L. S. Shapley. A Note On The Lemke Howson Algorithm. *Mathematical Programming Study 1: Pivoting and Extensions*, 1974.

[SS73] R.D. Smallwood and E.J. Sondik. The Optimal Control Of Partially Observable Markov Processes Over A Finite Horizon. *Operations Research*, 21(5):1071 – 1088, 1973.

[SZ07] Sven Seuken and Shlomo Zilberstein. Memory-bounded Dynamic Programming For DEC-POMDPs. *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2009 – 2015, 2007.

[TA85] J. Tsitsiklis and M. Athans. On The Complexity Of Decentralized Decision Making And Detection Problems. *IEEE Transactions on Automatic Control*, 30(5):440–446, 1985.

[VNTY06] Pradeep Varakantham, Ranjit Nair, Milind Tambe, and Makoto Yokoo. Winning Back the Cup for Distributed POMDPs: Planning Over Continuous Belief Spaces. *In Proceedings of the Fifth International Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2006)*, 2006.

[vS96] Bernhard von Stengel. Efficient Computation of Behavior Strategies. *Games and Economic Behavior*, 14:220–246, 1996.

[vS02] Bernhard von Stengel. Computing Equilibria Of Two-Person Games. *Handbook of Game Theory, Chapter 45 (R.J. Aumann and S. Hart, editors)*, 3:1723–1759, 2002.

[vSF06] Bernhard von Stengel and Françoise Forges. Extensive Form Correlated Equilibrium: Definition and Computational Complexity. *CDAM Research Report LSE-CDAM-2006-04*, 2006.

[Whi93] D. J. White. A Survey Of Applications Of Markov Decision Processes. *The Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

[Wil72] Robert Wilson. Computing Equilibria of Two-Person Games from the Extensive Form. *Management Science*, 18(7):448–460, 1972.

Index