DIPLOMARBEIT

XCS in dynamischen Multiagenten-Überwachungsszenarios ohne globaler Kommunikation

von

Clemens Lode

Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Universität Karlsruhe

Referent: Prof. Dr. Hartmut Schmeck
Betreuer: Dipl. Wi-Inf. Urban Richter

Karlsruhe, 30.03.2009

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Die Diplomarbeit bestand im Wesentlichen aus drei Teilen: Ausarbeitung Szenarien, Sensorik

1.1 How to make a reference

The conclusion can be found in Chapter 9.

1.2 How to cite

An interesting discussion can be found in [**?**]

1.3 How to add a figure

Of course there are plenty of other ways to add figures.

Abbildung 1.1: Caption shown below the figure.

Kapitel 2

Szenario

Im Wesentlichen sollen die hier besprochenen Algorithmen in einem überwachungsszenario getestet werden, d.h. die Qualität eines Algorithmus wird anhand des Anteils der Zeit bewertet, die er das Zielobjekt überwachen konnte, relativ zur Gesamtzeit.

Verwendetes Umfeld wird ein quadratischer Torus sein, der aus quadratischen Feldern besteht. Jedes bewegliche Objekt auf einem Feld kann sich nur auf eines der vier Nachbarfelder bewegen oder stehenbleiben. Die Felder können entweder leer oder durch ein Objekt besetzt sein. Besetzte Felder können nicht betreten werden. Es gibt drei verschiedene Arten von Objekten, unbewegliche Hindernisse, ein zu überwachende Zielagent und Agenten. Zielagent und Agent bewegen sich anhand eines bestimmten Algorithmus anhand bestimmter Sensordaten.

TODO Auflösen

2.1 Problem-Instanzen

Eine einzelne Probleminstanz entspricht einem Torus mit einer (abhängig vom verwendeten Random-Seed-Wert) bestimmten Anfangsbelegung mit bestimmten Objekten und bestimmten Parametern zur Sichtbarkeit. Die Parameter bestimmen, ob und wie Objekte

andere Objekte die Sicht versperren und bis zu welcher Distanz der Zielagent von einem Agenten als überwacht gilt, sofern die Sicht durch andere Objekte nicht versperrt ist.

Ein Experiment entspricht dem Test einer Anzahl von Probleminstanzen mit einer Reihe von Random-Seed-Werten. In einem Durchlauf werden mehrere Experimente (jeweils mit unterschiedlichen Random-Seed-Wert-Reihen) durchgeführt.

2.2 Qualität

Problem-Qualität eines Algorithmus zu einem Problem wird anhand des Anteils der Zeit berechnet, die er das Zielobjekt während des Problems überwachen konnte, relativ zur Gesamtzeit. Experiment-Qualität eines Algorithmus zu einer Anzahl von Problemen (einem Experiment) wird Anhand des Gesamtanteilder Zeit berechnet, die er das Zielobjekt während aller Probleme überwachen konnte, relativ zur Gesamtzeit aller Probleme. Qualität eines Algorithmus entspricht dem Durchschnitt aller Experimentqualitäten des Algorithmus.

Halbzeit-Problem-Qualität eines Algorithmus zu einem Problem entspricht dem Anteil der Zeit, die der Algorithmus das Zielobjekt während jeweils der zweiten Hälfte des Problems überwachen konnte, relativ zur halben Gesamtzeit. Halbzeit-Experiment-Qualität eines Algorithmus zu einer Anzahl von Problemen entspricht dem Anteil der Zeit, die der Algorithmus das Zielobjekt während jeweils der zweiten Hälfte des Problems überwachen konnte, relativ zur halben Gesamtzeit aller Probleme. Die Halbzeitqualität eines Algorithmus entspricht dem Durchschnitt aller Halbzeit-Experiment-Qualitäten eines Algorithmus.

Kapitel 3

Agenten

3.1 Sensoren

Jeder Agent besitzt 3 Gruppen mit jeweils 4 Binärsensoren. Alle Sensoren sind visuelle Sensoren mit begrenzter Reichweite. Je nach Parameter des Szenarios wird die Sicht durch andere Objekten blockiert. Sichtlinien werden durch einen einfachen Bresenham-Linienalgorithmus bestimmt. Jede Gruppe von Sensoren nimmt einen anderen Typ von Objekt wahr. Die erste Gruppe nimmt den Zielagenten, die zweite andere Agenten und die dritte Hindernisse. Ein Sensor ist jeweils in eine bestimmte Richtung ausgerichtet (Norden, Osten, Süden, Westen) und wird auf "wahr" gesetzt, wenn sich in dem von der Sichtweite bestimmten Kegel ein entsprechendes Objekt befindet.

3.2 Fähigkeiten

Jeder Agent kann in jedem Schritt zwischen 5 verschiedenen Aktionen wählen, die den vier Richtungen plus einer Aktion, bei der der Agent sich nicht bewegt, entsprechen. Agenten können pro Zeiteinheit genau einen Schritt durchführen. Der Zielagent kann je nach Szenario mehrere Schritte ausführen. Könnte der Zielagent ebenfalls nur einen

Schritt ausführen, wäre das Problem sehr simpel, da der Zielagent Schwierigkeiten hätte, Agenten abzuschütteln, deren einzige Strategie es ist, sich in Richtung des Zielagenten zu bewegen.

3.3 Ablauf der Bewegung

Alle Agenten werden nacheinander in der Art abgearbeitet, dass der jeweilige Agent die aktuellen Sensordaten aus der Umgebung holt und anhand dieser die nächste Aktion bestimmt. Ungültige Aktionen, d.h. der Versuch sich auf ein besetztes Feld zu bewegen, schlagen fehl und der Agent führt in diesem Schritt keine Aktion aus, wird aber nicht weiter bestraft. Eine detaillierte Beschreibung wird weiter unten geliefert ("Ablauf der Simulation").

3.4 Grundsätzliche Algorithmen der Agenten

Neben denjenigen Algorithmen die auf LCS basieren und weiter untern besprochen werden, gibt es folgende Grundtypen, die dazu dienen, die Qualität der anderen Algorithmen einzuordnen. Wesentliches Merkmal im Vergleich zu auf LCS basierenden Algorithmen ist, dass sie statische Regeln benutzen und den Erfolg oder Misserfolg ihrer Aktionen ignorieren.

3.4.1 "Randomized"

In jedem Schritt wird eine zufällige Aktion ausgeführt.

3.4.2 "Simple AI Agent"

Ist der Zielagent in Sichtweite, bewegt sich dieser Agent auf das Ziel zu. Ist das Ziel nicht in Sichtweite, führt er eine zufällige Aktion aus.

3.4.3 "Intelligent AI Agent"

Ist der Zielagent in Sicht, verhlt sich dieser Algorithmus wie "Simple AI Agent". Ist der Zielagent dagegen nicht in Sicht, wird versucht anderen Agenten auszuweichen um ein möglichst breit gestreutes Netz aus Agenten aufzubauen. In der Implementation heit das, dass unter allen Richtungen, in denen kein anderer Agent geortet wurde, eine zufällig ausgewählt wird. Falls alle Richtungen belegt (oder alle frei) sind, aus allen Richtungen eine zufällig ausgewählt wird.

3.5 Grundsätzliche Zielagententypen

Die Art der Bewegung des Zielagenten trägt grundsätzlich zur Schwierigkeit eines Szenarios bei. Gemeinsam haben alle Arten von Bewegungen des Zielagenten, dass, wenn kein freies Feld zur Verfügung steht, springt der Zielagent auf ein zufälliges freies Feld auf dem Grid springt. Dies kommt einem Neustart gleich und ist notwendig um einer Verfälschung des Ergebnisses zu verhindern, das dadurch rühren kann, dass ein oder mehrere Agenten (zusammen mit eventuellen Hindernissen) bis zum Ende des Problems alle vier Bewegungsrichtungen des Zielagenten blockieren.

3.5.1 "Random Movement"

Wie "Randomized". Sind alle möglichen Felder belegt, wird wie oben beschrieben auf ein zufälliges Feld gesprungen.

3.5.2 “Total Random”

Der Zielagent springt zu einem zufälligen (freien) Feld auf dem Grid. Mit dieser Einstellung kann die Abdeckung des Algorithmus geprüft werden, d.h. inwieweit die Agenten jeweils auerhalb der Überwachungsreichweite anderer Agenten bleiben. Jegliche Anpassung an die Bewegung des Zielagenten ist hier wenig hilfreich, ein Agent kann nicht einmal davon ausgehen, dass sich der Zielagent in der Nähe seiner Position der letzten Zeiteinheit befindet.

3.5.3 “One direction change”

Mit dieser Einstellung wird die der letzten Richtung entgegengesetzten Richtung aus der Menge der Auswahlmöglichkeiten entfernt und von den verbleibenden drei Richtungen (plus der Aktion “Stehenbleiben”) eine zufällig ausgewählt. Sind alle drei Richtungen versperrt, wird stehengeblieben. War die letzte Aktion nicht eine Bewegungsrichtung sondern die Aktion “Stehenbleiben”, so wird eine zufällige Richtung ausgewählt. Sind alle Richtungen versperrt wird auch hier wie bei “Random Movement” auf ein zufälliges Feld gesprungen.

3.5.4 “Always same direction”

Der Zielagent versucht immer Richtung Norden zu gehen. Ist das Zielfeld blockiert, wählt er ein zufälliges, angrenzendes, freies Feld im Westen oder Osten. Sind auch diese belegt, springt er wie oben auf ein zufälliges freies Feld. Schafft es der Zielagent innerhalb von einer bestimmten Zahl (Breite des Spielfelds) von Schritten nicht, einen weiteren Schritt nach Norden zu gehen, wird ebenfalls gesprungen um ein Festhängen an einem Hindernis zu vermeiden.

Abbildung 3.1: Zielagent macht pro Schritt maximal eine Richtungsänderung

3.5.5 "Intelligent (Open)"

Der Zielagent versucht andere Agenten zu vermeiden. Bei der Wahl der Richtung werden alle Richtungen gestrichen, in denen sich ein anderer Agent befindet. Von den verbleibenden Richtungen werden mit 20% Wahrscheinlichkeit alle Richtungen gestrichen, in denen sich ein Hindernis befindet. Sind alle Richtungen gestrichen worden, bewegt der Zielagent sich zufällig. Sind alle Richtungen blockiert, springt er wie in den anderen Einstellungen auch.

3.5.6 "Intelligent (Hide)"

Der Zielagent vermeidet andere Agenten wie bei "Intelligent (Open)", streicht aber statt Richtungen mit Hindernissen Richtungen ohne Hindernisse mit 20

3.5.7 "LCS"

Eine LCS Implementierung. Das Ziel des Zielagenten ist es hier aber, möglichst nicht andere Agenten zu überwachen (bzw. umgekehrt sich von anderen nicht überwachen zu lassen, d.h. nicht in die Überwachungsreichweite anderer Agenten zu geraten). Eine genaue Beschreibung folgt weiter unten.

Kapitel 4

Szenarien

Getestet werden eine Reihe von Szenarien (in Verbindung mit unterschiedlichen Werte für die Anzahl der Agenten, Gröe des Spielfelds und Art und Geschwindigkeit der Zielagentenbewegung). Wesentliche Rolle spielt hier die Verteilung der Hindernisse.

4.1 Random Scenario

Zwei Parameter bestimmen, wie das zufällige Szenario auszusehen hat, zum einen der Prozentsatz an Hindernissen an der Gesamtzahl der Felder, zum anderen der Grad inwieweit die Hindernisse zusammenhängen. Dieser Grad bestimmt nach Setzen eines Hindernisses die Wahrscheinlichkeit für jedes einzelne angrenzende freie Feld, dass dort sofort ein weiteres Hindernis gesetzt wird. Ein Wert von 0.0 ergibt somit ein völlig zufällig verteilte Menge an Hindernissen während ein Wert nahe eine oder mehrere stark zusammenhängende Strukturen schafft. Wird der Prozentsatz an Hindernissen auf Null gesetzt, dann werden Hindernisse vollständig deaktiviert. Als Optimierung werden in diesem Fall auch alle Sensorinformationen diesbezüglich deaktiviert. TODO optional

4.2 Pillar Scenario

Hier werden mit jeweils 7 Feldern Zwischenraum zwischen den Hindernissen (und mindestens 3 Feldern Zwischenraum zwischen Rand und den Hindernissen) Hindernisse verteilt. Idee ist, dass die Agenten eine kleine Orientierungshilfe besitzen aber gleichzeitig möglichst wenig Hindernisse verteilt werden. TODO Der Zielagent startet in der Mitte.

4.3 Cross Scenario

Hier gibt es in der Mitte eine horizontale Reihe aus Hindernissen halber Gesamtbreite welche durch eine vertikale Reihe aus Hindernissen halber Gesamthöhe geschnitten wird.

4.4 Room scenario

In der Mitte des Grids wird ein Rechteck der halben Gesamthöhe und -breite des Grids erstellt, welches eine öffnung von 4 Feldern Breite aufweist. Der Zielagent startet wie im Pillar Scenario in der Mitte, alle Agenten starten am Rand des Grids.

4.5 Difficult Scenario

Hier wird das Grid zum einen an der rechten Seite vollständig durch Hindernisse blockiert um den Torus zu halbieren. Alle Agenten starten am linken Rand, der Zielagent startet auf der rechten Seite. In regelmäigen Abständen (7 Felder Zwischenraum) befindet sich eine vertikale Reihe von Hindernissen mit Öffnungen von 4 Feldern Breite abwechselnd im oberen Viertel und dem unteren Viertel.

TODO vielleicht Agentenlösungen eines Szenarios in das andere einsetzen?

Kapitel 5

Ablauf der Simulation

Bei Simulationen am Computer stellt sich sofort die Frage nach der Genauigkeit. Die Agenten werden bei dieser Simulation nacheinander abgearbeitet und bewegen sich auf einem diskreten Grid. Dies kann u.U. dazu führen, dass je nach Position in der Liste abzuarbeitender Agenten die Informationen über die Umgebung unterschiedlich alt sind. Die groe Frage ist deshalb, in welcher Reihenfolge Sensordaten ermittelt, ausgewertet, Agenten bewegt, intern sich selbst bewertet und global die Qualität gemessen wird. Einzige Restriktionen sind, dass eine Aktion nach der Verarbeitung der Sensordaten stattfinden muss und eine Bewertung einer Aktion nach dessen Ausführung stattfinden muss. Ansonsten gibt es folgende Möglichkeiten:

1. Für alle Agenten werden erst einmal die neuen Sensordaten erfasst und sich für eine Aktion entschieden. Sind alle Agenten abgearbeitet werden die Aktionen ausgeführt.

2. Die Agenten werden nacheinander abgearbeitet, es werden jeweils neue Sensordaten erfasst und sich sofort für eine neue Aktion entschieden.

Bei der ersten Möglichkeit haben alle Agenten die Sensordaten vom Beginn der Zeiteinheit, während bei der zweiten Möglichkeit später verarbeitetere Agenten bereits die

Aktionen der bereits berechneten Agenten miteinbeziehen können. Umgekehrt können dann frühere Agenten bessere Positionen früher besetzen. Da aufgrund der primitiven Sensoren nicht davon auszugehen ist, dass Agenten beginnende Bewegungen (und somit deren Zielposition) anderer Agenten einbeziehen können, soll jeder Agent von den Sensorinformationen zu Beginn der Zeiteinheit ausgehen. Die Reihenfolge der Ausführung der Aktionen spielt eine Rolle, wenn mehrere Agenten sich auf das selbe Feld bewegen wollen. Arbeiten wir die Liste unserer Agenten einfach linear ab, haben vorne stehende Agenten eine höhere Wahrscheinlichkeit, dass ihre Aktion ausgeführt wird. Da es keine Veranlassung gibt, ihnen diesen Vorteil zu geben, werden Aktionen in zufälliger Reihenfolge abgearbeitet. Bezüglich der Bewegung ergibt sich hierbei eine weitere Frage, nämlich wie unterschiedliche Bewegungsgeschwindigkeiten behandelt werden sollen. Alle Agenten haben eine Einheitsgeschwindigkeit von maximal einem Feld pro Zeiteinheit, während sich der Zielagent je nach Szenario gleich eine ganze Anzahl von Feldern bewegen kann. Auch hier habe ich mich für eine zufällige Verteilung entschieden. Kann sich der Zielagent um n Schritte bewegen, so wird seine Bewegung in n Einzelschritte unterteilt, die nacheinander mit zufälligen Abständen (d.h. Bewegungen anderer Agenten) ausgeführt werden. Eine weitere Frage ist, wie der Zielagent diese weiteren Schritte festlegen soll. Hier soll ein Sonderfall eingeführt, so dass der Zielagent in einer Zeiteinheit mehrmals (n-mal) neue Sensordaten erfassen und sich für eine neue Aktion entscheiden kann.

Schlielich bleibt die Frage danach, wann geprüft werden soll, ob der Zielagent in Sicht ist, und wann somit der Reward verteilt wird. Da XCS in der Standardimplementation darauf ausgelegt ist, den Reward jeweils genau einer Aktion zuzuordnen, sollte der Reward nicht bei jeder einzelnen Bewegung des Zielagenten überprüft und verteilt werden, sondern nur einmal pro Zeiteinheit. Auerdem soll der Einfachheit halber der Reward auch von binärer Natur sein (Zielagent in überwachungsreichweite oder Zielagent nicht in überwachungsreichweite), weshalb Zwischenzustände für den Reward (z.B. War zwei von

drei Zeitschritten in der überwachungsreichweite =¿ 2/3 Reward) ausgeschlossen werden sollen. Für den Reward gibt es somit folgende Möglichkeiten:

1. Rewards werden direkt nach der Ausführung einer einzelnen Aktion vergeben
2. Rewards werden nach Ausführung aller Aktionen der Agenten vergeben
3. Rewards werden nach Ausführung aller Aktionen des Zielagenten vergeben

Werden die Rewards sofort vergeben (Punkt 1), dann werden sich später noch wegbewegende bzw. sich später noch hin-bewegende Agenten nicht beachtet. Da die Agenten in zufälliger Reihenfolge abgearbeitet werden, würde das bedeuten, dass die Bewegung der restlichen (zufälligen) Anzahl von Agenten in den Reward nicht miteinbezogen wird. Selbiges gilt für Punkt 3. Auch der Zielagent kann Reward erhalten. Hierbei gibt es ebenfalls drei Möglichkeiten:

1. Reward direkt nach Ausführung des letzten Schritts
2. Reward nach Ausführung aller Agenten

Eine konkrete Antwort kann man auf diese zwei Fragen nicht geben, sie hängt nämlich davon ab, was man denn nun eigentlich erreichen möchte, also auf welche Weise die Qualität des Algorithmus bewertet wird. Der naheliegendste Messzeitpunkt ist nachdem sich alle Agenten bewegt haben. Da wir Agenten und Zielagenten in einem Durchlauf gemeinsam bewegen, stellt sich die Frage nicht, ob wir womöglich vor der Bewegung des Zielagenten die Qualität messen sollen. Eine Messung nach der Bewegung des Zielagenten würde diesem erlauben, sich vor jeder Messung optimal zu positionieren, was in einer geringeren Qualität für den Algorithmus resultiert, da sich der Zielagent aus der Überwachungsreichweite anderer Agenten hinausbewegen kann. Letztlich ist es eine Frage der Problemstellung, denn eine Messung nach Bewegung des Zielagenten bedeutet letztlich, dass ein Agent, einen gerade aus seiner Überwachungsreichweite heraus laufenden

Zielagenten in diesem Schritt nicht mehr überwachen kann. Da ein wesentlicher Bestandteil die Kooperation (und somit die Abdeckung des Spielfelds anstatt dem Verfolgen des Zielagenten) sein soll, soll ein Bewertungskriterium sein, inwieweit der Einfluss des Zielagenten minimiert werden soll. Auch findet, wenn wir vom realistischen Fall ausgehen, die Bewegung des Zielagenten gleichzeitig mit allen anderen Agenten statt. Die Qualität wird somit nach der Bewegung des Zielagenten gemessen. Die Überlegung unterstreicht auch nochmal, dass es besser ist, den Zielagenten insgesamt wie einen normalen (aber sich mehrmals bewegenden) Agenten zu behandeln. Was den Zeitpunkt des Rewards betrifft, lautet die Hypothese, dass wir ein besseres Ergebnis erreichen, wenn man den Reward anhand der selben Momentaufnahme verteilt, anhand der wir auch die Qualität testen, d.h. dass Punkt 2 Punkt 1 überlegen ist. Dies bestätigt sich in Tests siehe TODO.

Zusammenfassend sieht der Ablauf aller Agenten (inklusive des Zielagenten) also wie folgt aus:

1. Erfassung aller Sensordaten
2. Wahl der Aktion anhand der Regeln des jeweiligen Agenten
3. Ausführung der Aktion (in zufälliger Reihenfolge, der Zielagent führt nach dem ersten Schritt auerdem Schritt 1. und 2. für alle weiteren Schritte nochmals durch)
4. Bestimmung des Rewards
5. Bestimmen der Qualität dieser Zeiteinheit

Kapitel 6

LCS

Jeder Agent besitzt ein Learning Classifier System. Das LCS besteht aus einer Reihe von Classifiern.

TODO anfaengliche Initialisierung?

6.1 Classifier

Ein Classifier besteht im Wesentlichen aus fünf Teilen:

6.1.1 Fitness

Der Fitness Wert soll die allgemeine Genauigkeit des Classifiers repräsentieren. TODO Wilson. Der Wertebereich verläuft zwischen 0.0 und 1.0 (maximale Genauigkeit).

6.1.2 Prediction

Der "Prediction"-Wert des Classifiers stellt die Höhe des Rewards dar, von dem der Classifier vermutet, dass er ihn bei der nächsten Vergabe des Rewards erhalten wird.

6.1.3 Prediction Error

Der Prediction-Error-Wert soll die Genauigkeit des Classifiers bzgl. der Reward-Prediction (durchschnittliche Differenz zwischen Prediction und tatsächlichem Reward) repräsentieren.

6.1.4 Aktion

Wird ein Classifier ausgewählt, wird eine bestimmte Aktion ausgeführt. In unserem Szenario entsprechen die Aktionsmöglichkeiten die der anderen Agenten, also 4 Bewegungsrichtungen plus eine "nichts-tun"-Aktion.

6.1.5 Kondition

Die Kondition gibt an, bei welchem Sensor-Input dieser Classifier ausgewählt werden kann.

6.2 Sensoren und Matching

In der hier verwendeten Implementierung gibt es zwar eine gewisse Vorverarbeitung der Sensordaten, im Wesentlichen müssen aber Kondition und Sensordaten übereinstimmen, damit der Classifier ausgewählt wird. Konkret besteht die Kondition ebenfalls aus einem 9-stelligen Vektor, der allerdings nicht nur binäre, sondern auch trinäre Werte besitzen kann.

6.2.1 Wildcards

Neben den zu den Sensordaten korrespondierenden Werten 0 und 1 gibt es noch einen dritten "dont-care"-Zustand "#", der anzeigen soll, dass beim Vergleich zwischen Kondition und Sensordaten diese Stelle ignoriert werden soll. Eine nur aus "dont-care" Werten

bestehende Kondition würde somit bei der Auswahl immer in Betracht gezogen werden, da er auf alle Sensor-Inputs passt.

Beispiel: Kondition 1.#010.1#01 matched Sensordaten 1.0010.1001, 1.1010.1001, 1.0010.1101 und 1.1010.1101.

Die Benutzung von Wildcards erlaubt es dem LCS mehrere Classifier zu subsummieren, wodurch die Gesamtzahl der Classifier sinkt und somit die Erfahrungen, die ein LCS Agent sammelt, nicht unbedingt doppelt gemacht werden müssen.

6.3 Drehungen

Ein Classifier besteht aus dem Bedingungsvektor

$$(gx_0x_1x_2x_3)$$

(bzw.

$$(gx_0x_1x_2x_3y_0y_1y_2y_3)$$

für den Fall mit Hindernissen) und der Aktion a.

Eine wesentliche Vereinfachung, die angenommen wird, ist, dass angenommen wird, dass eine Aktion in einer anderen, in 90 Grad Schritten gedrehten Umwelt, gleiche Güte besitzt. In einer statischen Umgebung ist dies nicht unbedingt der Fall, ohne diese Vereinfachung könnte sich ein Agent einfacher zurechtfinden, wie TODO (keine Drehung, 1 Problem pro Experiment) diese Testläufe zeigen. Da wir uns aber auf den dynamischen Fall konzentrieren und durch diese Vereinfachung eine signifikante Verkleinerung des Suchraums erreichen, benutzen wir die Vereinfachung.

Im Algorithmus betrifft dies primär den Vergleich von Classifiern untereinander und mit dem Sensorstatus. Ein Classifier A ist identisch mit einem Classifier B wenn gilt:

1.

$$A_g = B_g$$

2. Falls

$$A_g = 0$$

:

   (a) Es gibt ein i für das gilt:

$$(A_{x_{0+i \bmod 4}} A_{x_{1+i \bmod 4}} A_{x_{2+i \bmod 4}} A_{x_{3+i \bmod 4}}) = (B_{x_0} B_{x_1} B_{x_2} B_{x_3})$$

   (b) Mit Hindernissen muss für dieses i auerdem gelten:

      i.

$$(A_{y_{0+i \bmod 4}} A_{y_{1+i \bmod 4}} A_{y_{2+i \bmod 4}} A_{y_{3+i \bmod 4}}) = (B_{y_0} B_{y_1} B_{y_2} B_{y_3}$$

      ii. Falls $A_a = \mathrm{NO_ACTION}$:

$$A_a = B_a$$

      iii. Falls $A_a \neq \mathrm{NO_ACTION}$:

$$(A_a + i) \bmod 4 = B_a$$

3. Falls $A_g = 1$:

   (a)

$$(A_{x_0} A_{x_1} A_{x_2} A_{x_3}) = (B_{x_0} B_{x_1} B_{x_2} B_{x_3})$$

   (b) Mit Hindernisse muss auerdem gelten:

$$(A_{y_0} A_{y_1} A_{y_2} A_{y_3}) = (B_{y_0} B_{y_1} B_{y_2} B_{y_3})$$

   (c)

$$A_a = B_a$$

Die Gleichheit zwischen Vektoren gilt, wenn paarweise Gleichheit zwischen den Elementen besteht, also $A_{x_i} = B_{x_i}$ für $i = 0 \ldots 3$ und $A_{y_i} = B_{y_i}$ für $i = 0 \ldots 3$ im Fall mit Hindernissen.

Beim Vergleich mit Sensordaten wir ebenfalls mit einem Vektor wie bei B erglichen. Entscheidend beim Vergleich ist hier aber nicht, dass beide Vektoren identisch sind, sondern, dass der Classifier "matched". Ein Element des Bedingungsvektors kann 3 Zustände einnehmen. 0, 1 und #. # beinhaltet beide Zustände 0 und 1.

Den dritten verwendeten Vergleich zwischen Bedingungsvektoren gibt es bei der Subsumation (der Kinder zu ihren Eltern und des gesamten ActionSets siehe TODO). Ein Classifier subsumiert einen anderen Classifier, wenn er ihn beinhaltet, aber nicht identisch mit ihm ist, also allgemeiner ist.

Die Drehung spielt auerdem eine Rolle beim "Covering". In der Originalversion von XCS kann aus einem Classifier die tatsächliche (phänotypische) Aktion direkt abgelesen werden. In dieser neuen Version mit Drehung muss für jeden Classifier alle möglichen Drehungen bestimmt werden. Ein einzelner Classifier kann mehrere Aktionen abdecken, beispielsweise 0-0000-0 etc. TODO

Für die Auswahl der tatsächlich auszuführenden Aktion müssen dann alle (genotypischen) Classifier in AppliedClassifier umgewandelt werden, die jeweils genau auf eine bestimmte Aktion verweisen. Sichtwe

6.4 Ablauf

1. Bei der Auswahl einer Aktion werden zuerst einmal alle Classifier mit denjenigen Konditionen gesucht, die auf die aktuellen Sensordaten passen. Diese bilden dann das MatchSet.

2. Im nächsten Schritt wählen wir einen Classifier aus diesem MatchSet aus und spei-

chern dessen Aktion.

3. Schlielich bilden wir anhand des MatchSets und der gewählten Aktion das ActionSet

6.4.1 Exploration and Exploitation

Die Auswahl in Punkt 2 hängt von einem Szenarioparameter ab, der im wesentlichen unterscheidet, ob entweder der beste Classifier gewählt wird ("exploit") oder ein zufälliger anhand der roulette selection ("explore"). Qualität eines Classifiers soll durch das Produkt aus Fitness * Prediction sein. Hierbei gibt es 5 verschiedene Möglichkeiten:

1. Immer "exploit"
2. Immer "explore"
3. Abwechselnd "explore" und "exploit"
4. Zufällig entweder "explore" oder "exploit" (50% Wahrscheinlichkeit jeweils)
5. Erste Hälfte eines jeden Problems nur "explore", dann nur "exploit"
6. Wie (4.), während des Problems allerdings eine lineare Abnahme der Wahrscheinlichkeit von "explore" und eine lineare Zunahme der Wahrscheinlichkeit von "exploit"
7. "exploit" wenn Ziel in Sichtweite, "explore" sonst

Möglichkeit (3.) entspricht dem Fall in der Standardimplementierung von XCS. Dabei wird bei jedem Erreichen eines positiven Rewards zwischen "explore" und "exploit" hin und hergeschaltet, was in der Standardimplementierung dem Beginn eines neuen Problems entspricht.

TODO Vergleich

TODO Kommunikation

Wurde die Aktion ausgewählt und später schlielich ausgeführt, kommen wir nun zur Rewardvergabe.

6.5 Implementation

Als erster Schritt wird der aus der Literatur bekannte Algorithmus XCS näher untersucht. In der Literatur gibt es keine bekannte Implementierung von XCS in einem überwachungsszenario dieser oder ähnlicher Art. TODO

Alle Agenten haben dabei ein eigenes, unabhängiges ClassifierSet und können keine Regeln untereinander austauschen. Jeder Agent muss also selbst lernen.

Die Implementierung entspricht der Standardimplementation von Butz 2000, eine wesentliche Besonderheit stellt allerdings die Problemdefinition dar. Im Multistepverfahren läuft ein Problem so lange bis ein positiver Reward aufgetreten ist. Dann wird das selbe Szenario neu gestartet. Bei einem überwachungsszenario ist dies nicht möglich, das Szenario kann nicht neugestartet werden. Ziel ist hier ja nicht, einen bestimmten Weg zu einem festen Ziel zu finden (wie z.B. bei WOODS TODO), sondern eine bestimmte Regelmenge zu erlernen, mit der eine möglichst gute, dauerhafte überwachung stattfinden kann. In der Implementierung hier läuft das Problem deshalb einfach weiter. In jedem Schritt, in dem der Agent den Zielagenten sieht, wird also ein neues Multistep-Problem gestartet. TODO

6.5.1 Verwendeter Algorithmus: XCS

Bei einem überwachungsszenario mit kontinuierlichem Reward ist das in der Literatur (TODO Literaturverweis) Multistepverfahren nicht anwendbar. Desweiteren fehlen Arbeiten, in denen untersucht wird, inwieweit Kooperation zwischen Agenten (ohne globale Steuerungseinheit, ohne Regelaustausch) in diesem Zusammenhang angewendet wird. Die-

se Arbeit soll diese Lücke schlieen und die Basis für weitere Arbeiten in dieser Richtung liefern.

6.5.2 Wesentliche Veränderungen zu XCS:

ActionSets werden über eine ganze Probleminstanz hinweg gespeichert. Rewards werden direkt rückwirkend vergeben, sobald ein Event eintritt.

Weitere Anpassung: Ähnlichkeit von Classifiern über die Drehung und Aktion EA wenig Einflu, Crossing over lediglich auf 2 Stellen (Zielagent, Hindernisse)

Definition Event: Rewardänderung (Zielagent in Sicht ? Zielagent nicht in Sicht)

Wesentliches Gütekriterium war die Zahl der Zeitschritte, in denen der Zielagent sich in Reward-Reichweite eines beliebigen Agenten befand, geteilt durch die Gesamtzahl der Zeitschritte. Zeitschritte in denen gelernt wird, d.h. Exploration betrieben wird, werden nicht gesondert behandelt.

Zur Durchführung und Forschung war es notwendig, den kompletten XCS Algorithmus nachvollziehen zu können. TODO

Besonders die Verwaltung der Numerosity und die Verwendung des maxPrediction TODO

Das Multistepverfahren baut darauf auf, dass die Qualität der Agenten sich sukzessive mit jeder Probleminstanz verbessert, der Reward eben an immer weiter vom Ziel entfernte Aktionen TODO weitergereicht wird.

Der hier entwickelte Algorithmus muss primär nicht einen Weg zum Ziel erkennen, sondern eine möglichst optimale (und auch an andere Agenten angepasste) Verhaltensstrategie finden.

Da sich das Ziel schneller bewegt, kann eine einfache Verfolgungsstrategie nicht zum Erfolg führen. Eine einfache Implementation mit einem simplen Agenten der auf das Ziel zugeht, wenn es in Sicht ist und sich sonst wie ein sich zufällig bewegender Agent verhält,

schneidet grundsätzlich schlechter ab.

6.5.3 Numerosity

TODO Beschreibung In der originalen Implementierung von Butz 2000 TODO war die Behandlung der Numerosity stark optimiert auf den Fall des einmaligen Rewards ohne Protokollierung der bisherigen ActionSets. Nach einer missglückten ersten Implementierung der Wert der numerositySum eines ClassifierSets stimmte nicht mehr mit der Summe der numerosity-Werte der enthaltenen Classifiers überein entschloss ich mich den entsprechenden Code neuzuschreiben. Hierbei wurde jedem Classifier eine Liste der Eltern, d.h. der jeweiligen ActionSets, zugewiesen. Wird ein Microclassifier entfernt, wird dann lediglich die Änderungsfunktion der Numerosity des Classifiers aufgerufen, der dann wiederum die NumerositySum der jeweiligen Eltern anpasst. Dies macht einige Optimierungen rückgängig, erspart aber sehr viel Umstände, die NumerositySum immer auf den aktuellen Stand zu halten. Positiver Nebeneffekt ist, dass man dadurch leicht auf die Menge der ActionSets zugreifen kann, denen ein Classifier angehört. Inwiefern das tatsächlich von Nutzen sein kann ist offen. TODO

Kernstück des LCS Agenten:

6.5.4 Covering:

Covering ist fast identisch mit der originalen Version von XCS.

In meiner ersten Version hatte ich alle ungültigen Aktionen von vornherein für das Covern ausgeschlossen. Eine ungültige Aktion ist beispielsweise ein Laufen gegen ein Hindernis oder einen Agenten. Alleine dadurch verbesserte sich die Leistung um etwa 0.5%. Das lässt sich darauf zurückführen, dass die Sensoren eines Agenten eigentlich nur feststellen können, ob ein anderer Agent in Sichtweite ist, nicht aber in welcher Entfernung. Für die eigentlichen Ergebnisse wurde die dafür verantwortliche Methode wieder entfernt.

Auerdem ist ein Fehler der originalen XCS Implementation behoben. Wenn neue Classifier beim Covering hinzugefügt werden, wird ihre anfängliche ActionSetSize auf die numerositySum des matchSets gesetzt. Einen Grund dafür gibt es nicht, in meiner Implementation setze ich deshalb den Wert auf die anfängliche Gröe des actionSets. Beim Aufruf des Covering-Algorithmus weiss man schon, wie gro

TODO nein!

In der Implementation habe ich den Vorgang in zwei Schritte aufgeteilt, zuerst wird überprüft, ob das momentane ClassifierSet alle möglichen Aktionen abdeckt, im zweiten Schritt baue ich dann separat das matchSet auf. Neben besserer Programmlogik (es ist nicht intuitiv, wenn ein Konstruktor die übergebenen Parameter verändert) ist es auch übersichtlicher. TODO

6.5.5 Subsummation

Ein Problem ist hier natürlich die Sicherstellung, dass Informationen nicht verloren gehen. Andere Arbeiten befassen sich mit der Untersuchung von der Benutzung von Wildcards. In der hier verwendeten Implementation übernehme ich unverändert die Implementation aus der Literatur.

6.5.6 Verteilung des Rewards

Im Rahmen der Diplomarbeit wurden drei wesentliche Arten der Rewardverteilung getestet.

Rewardfunktion: 1 wenn Zielagent in Reichweite dieses Agenten 0 wenn nicht

Multistepverfahren: Idee ist, dass der Reward, den eine Aktion (bzw. das jeweils zugehörige ActionSet) erhält, vom erwarteten Reward der folgenden Aktion abhängt. Somit wird, rückführend vom Ziel, der Reward schrittweise an vorgehende Aktionen verteilt und somit das Ziel schneller gefunden

TODO Quelle

reward = 0 ? Gebe maxPrediction des nächsten Zugs reward = 1 ? Gebe maxPrediction 0, reward = 1

In jedem Schritt wird das vorherige ActionSet durch maxPrediction bzw. reward belohnt

6.6 LCS Varianten

6.6.1 LCS Variante 1

Die Hypothese bei der Aufstellung des Algorithmus ist im Grunde die selbe wie beim einfachen Multistepverfahren, dass die Kombination mehrerer Aktionen zum Ziel führt. Die wesentliche Verbindung beim Multistepverfahren zwischen den ActionSets der einzelnen Zeitschritte war die zeitliche Nähe, ein einem hoch bewerteten ActionSet folgendem ActionSet wird ebenfalls hoch bewertet. Bei der veränderten LCS Variante ist die Verbindung zwischen den ActionSets direkt die zeitliche Nähe zu einem Ereignis. ActionSets von jedem Schritt werden gespeichert bis ein Event auftritt und dann in Abhängigkeit des Alters bewertet.

$$r(a)$$

bezeichnet den Reward für das ActionSet mit Alter

$$a$$

.

$$r(a) = \begin{cases} \mathrm{MAX_R EWARD} \frac{a}{\mathrm{size(ActionSet)}} \mathrm{fürreward} = 1 \\ \mathrm{MAX_R EWARD} \frac{1-a}{\mathrm{size(ActionSet)}} \mathrm{fürreward} = 0 \end{cases}$$

TODO Vergleich mit quadratischem Abstieg

Es wird auf ein Event gewartet. Ein Event ist eine Änderung des Rewards zwischen zwei Zeitschritten. Tritt ein solches Event auf, werden alle Aktionen seit der letzten Änderung absteigend belohnt bzw. bestraft.

Sonderfall: Es tritt für einen Agenten kein Event auf.

Eine Konstante “maxStackSize” bestimmt, wann das Warten auf ein neues Event abgebrochen werden soll. Im Fall eines Abbruchs wird die Hälfte des Stacks (also die ältesten Einträge) mit dem damals vergebenem Reward (welcher dem aktuellen Reward entspricht, es ist ja keine Rewardänderung, d.h. Ein Event, eingetreten) kreditiert und vom Stack genommen. Anschlieend wird normal weiter verfahren bis der Stack wieder voll ist bzw. bis ein Event auftritt.

Wesentliche Ergebnisse (Fall ohne Kommunikation)

LCS Agenten schneiden auch ohne Kommunikation (bei ausreichender Anzahl von Schritten) immer besser ab als zufällige Agenten.

¡Grafiken¿

Test verschiedener Exploration-Modi Bis auf wenige Ausnahmen (Liste) irrelevant ob man nun 50%/50%, absteigend o.ä. macht. Nur ausschlielich Exploration bzw. ausschlielich Exploitation können gewisse Schwankungen auftreten.

6.6.2 Evolutionärer Algorithmus

Der genetische Algorithmus wurde im Wesentlichen nicht verändert. Da aber die Binärsensoren eng zusammenhängen werden beim Crossing Over 2 feste Stellen für Crossing Over benutzt. Die Stellen markieren die 3 Gene (Zielagent, Agenten, feste Hindernisse).

Im Test erbrachte die Benutzung des Algorithmus wenig Unterschied.

Kapitel 7

Kommunikation

Da wir ein Multiagentensystem betrachten, stellt sich natürlich die Frage nach der Kommunikation. In der Literatur gibt es Multiagentensysteme die auf Learning Classifier Systemen aufbauen, wie z.B. TODO Literatur. Alle Ansätze in der Literatur erlauben jedoch globale Kommunikation, z.T. Gibt es globale Classifier auf die alle Agenten zurückgreifen können, z.T. gibt es globale Steuerung.

In dieser Arbeit betrachte ich das Szenario ohne globale Steuerung oder globale Classifier, also mit der Restriktion einer begrenzten, lokalen Kommunikation. Geht man davon aus, dass über die Zeit hinweg jeder Agent indirekt mit jedem anderen Agenten in Kontakt treten kann, Nachrichten also mit Zeitverzögerung weitergeleitet werden können, ist eine Form der globalen, wenn auch zeitverzögerten, Kommunikation möglich. TODO Eine spezielle Implementierung für diesen Fall werde ich weiter unten besprechen TODO

Ohne Kommunikation wird jeder Agent versuchen, den Agenten selbst möglichst in Sichtweite zu bekommen. Je schwieriger die Zielagentenbewegung nachzuvollziehen ist, also je zufälliger und schneller sie abläuft, desto wi

TODO Weitergabe an alle Agenten: Keinen tieferen Sinn. Wenn nicht anhand von Verhaltensmerkmalen diskriminiert wird, entspricht diese Form der Weitergabe des Re-

wards einer zufälligen Bewertung der Classifier anderer Agenten. In Tests hat sich gezeigt, dass dadurch in bestimmten Fällen deutlich bessere Ergebnisse erreicht werden können als im Fall ohne Kommunikation. Dies ist darauf zurückzuführen, dass in dem Fall die Kartengröe und Zielagentengeschwindigkeit relativ zur Sichtweite und Lerngeschwindigkeit zu gro war, die Agenten also annahmen, dass ihr Verhalten schlecht ist, weil sie den Zielagenten relativ selten in Sicht bekamen. Eine Weitergabe des Rewards an alle Agenten kann hier zu einer Verbesserung führen, dabei ist der Punkt aber nicht, dass Informationen ausgetauscht werden, sondern, dass obiges Verhältnis gedreht wird.

TODO Factor

Eine zweite Implementation berechnet erst einmal für jeden Agenten einen "Egoismus-Faktor", indem grob die Wahrscheinlichkeit ermittelt wird, dass ein Agent, wenn sich ein anderer Agent in Sicht befindet, sich in diese Richtung bewegt. "Egoismus"-Faktor, weil ein groer Faktor bedeutet, dass der Agent eher einen kleinen Abstand zu anderen Agenten bevorzugt, also wahrscheinlich eher auf eigene Faust versucht, den Zielagenten in Sicht zu bekommen und nicht kooperiert. TODO Die Hypothese ist, dass Agenten mit änhlichem Egoismus-Faktor auch einen ähnlichen Classifiersatz besitzen und der Reward nicht an alle Agenten gleichmäig weitergegeben wird, sondern bevorzugt an ähnliche Agenten. Damit gäbe es einen Druck in Richtung eines bestimmten Egoismus-Faktors. TODO

Der Kommunikationsaufwand ist hier nur minimal gröer, neben dem Reward muss der Faktor übertragen werden.

Eine dritte Implementation vergleicht die Classifiers direkt. Alle Classifier des Agenten, der den Reward weitergibt, die ausreichend Erfahrung gesammelt haben und ausreichend genau ist (Experience und geringes PredictionError, identisch mit isPossibleSubsumer), werden mit einem identischen Classifier (d.h. mit gleicher Condition und gleicher Action) verglichen. Die Differenz der Produkte aus Fitness und Prediction geteilt durch den gröeren Prediction-Wert der beiden Classifier stellt hier den Faktor dar.

TODO Formel

Weitere Implementationen sind denkbar, bei denen komplexere Vergleiche und Analysen durchgeführt werden. TODO

Durch eine gemeinsame Schnittstelle erhält jeder Agent dann den Reward zusammen mit dem Factor. Dabei ergibt sich das Problem, dass sich Rewards überschneiden können, da jeder Reward sich rückwirkend auf die vergangenen ActionClassifierSets auswirkt. Es können mehrere externe Rewards eintreffen als auch ein eigener lokaler Reward aufgetreten sein. Würden die Rewards nach ihrer Eingangsreihenfolge abgearbeitet, kann es passieren, dass das selbe ActionClassifierSet sowohl einen guten als auch einen schlechten Reward erhält. Dies macht aber wenig Sinn, da nur der beste Reward von Bedeutung ist. Befindet sich das Ziel in Überwachungsreichweite und verliert ein anderer Agent das Ziel aus der Sicht, sollte der Agent, der das Ziel in Sicht hat, deswegen nicht bestraft werden.

In Kapitel TODO wurden 4 verschiedene mögliche Situationen für einen einzelnen Agenten dargestellt.

Bezieht man den Zustand anderer Agenten mit ein, ergeben sich insgesamt 16 verschiedene Situationen, nämlich, dass keine Veränderung aufgetreten ist und der Zielagent sich in diesem wie auch im vorherigen Schritt in bzw. auer Überwachungsreichweite anderer Agenten befindet, oder, dass eine Veränderung (in Sicht ? nicht in Sicht bzw. nicht in Sicht ? in Sicht) aufgetreten ist.

Hier die erweiterte Übersicht, welche Arten von Events im Rahmen eines Multiagentensszenarios auftreten können:

Ziel befindet sich von anderen Agenten in Sicht: Time-out (Ziel in Sicht) Time-out (Ziel nicht in Sicht)

Ziel kommt in Sicht Ziel verschwindet aus Sicht

Gebe keinen Reward an andere Agenten weiter. Es ist nicht relevant, ob ein Agent das Ziel aus den Augen verliert oder nicht, es ist nur relevant, ob der Zielagent weiterhin

von anderen Agenten beobachtet wird. Ein Sonderfall ist, wenn im vorherigen Schritt der Zielagent nicht in Sichtweite eines anderen Agenten stand, also in diesem Schritt auf einmal mehrere Agenten den Zielagenten sehen können. In diesem Fall gibt nur der erste Agent den Reward weiter und setzt ein Flag.

Ziel befindet sich von anderen Agenten nicht in Sicht: Time-out (Ziel in Sicht) Time-out (Ziel nicht in Sicht)

Ziel verschwindet aus Sicht War der Zielagent von keinem anderen Agenten in Sicht, dann hat sich der Zielagent hiermit aus der Sichtweite aller Agenten bewegt. Somit haben alle Agenten versagt und der negative Reward wird weitergegeben.

Selbiges wenn das Ziel in Sicht kommt und von keinem anderen Agenten in Sicht ist. Die Agenten waren offensichtlich erfolgreich und können belohnt werden.

Kommunikation

Wann immer ein Reward an einen Agenten verteilt wird, kann es sinnvoll sein, diesen Reward an andere Agenten weiterzugeben. Im Rahmen der Diplomarbeit soll Kommunikationsreichweite und -bandbreite keine Rolle spielen. Mit diesen Restriktionen kann man erwarten, dass die Effektivität nicht viel geringer ist, sofern eine Kommunikationsreichweite gröer der Sichtweite gegeben ist, ein Agent (mit anderen Agenten als Proxy) mit fast jedem anderen Agenten in Kontakt treten kann.

Ist kein Event aufgetreten und leeren wir die Hälfte des Stacks ist es nicht sinnvoll, einen 0-Reward weiterzugeben, da zwangsläufig immer mehrere Agenten eine längere Zeit den Zielagenten nicht sehen, selbst wenn sie sich optimal verteilen / bewegen. TODO

Dies zeigt auch der Test: TODO

Ist kein Event aufgetreten und haben wir einen 1-Reward vorliegen, dann stellt sich die Frage, ob bereits andere Agenten diesen Reward weitergereicht haben. Befinden sich andere Agenten in Reichweite soll nur ein Agent den Reward weiterreichen. TODO Test

Bei einem Event geben wir den Reward weiter. Allerdings muss hier der Spezialfall

berücksichtigt werden, dass andere Agenten das Ziel ebenfalls in Sichtweite haben. Es gibt keinen Grund, weshalb Agenten n-fach bestraft werden, wenn der Zielagent sich aus dem Sichtfeld von n Agenten bewegt. Umgekehrt macht es keinen Sinn, dass andere Agenten dafür belohnt werden, dass der Zielagent mehrfach in Sicht kommt TODO.

Verzögerter Reward

Wird der Reward an andere Agenten verteilt, kann es dazu kommen, dass ActionSets mehrfach Reward erhalten.

7.1 Verwandtschaftsgrad

Vergleich reward-all reward simple

Bewertung Kommunikation:

Die Vorteile, die man durch Kommunikation erzielen kann, hängt stark durch das Szenario ab. Beispielsweise in dem Fall, bei dem zufällige Agenten bereits fast 100Umgekehrt ist der Einfluss bei sehr wenigen Agenten gering. TODO

Vergleich unterschiedliche Agentenanzahl, unterschiedliche Kommunikationsmittel Vergleich mit LCS?

Old LCS Agent New LCS Agent

Multistep LCS Agent Dieser Algorithmus stellt eine Implementation des Standard XCS Algorithmus dar. Unterschied zur Standardimplementation ist, dass die Probleminstanz bei Erreichen des temporären Ziels (d.h. den Zielagenten in Sicht zu bekommen) nicht tatsächlich neugestartet wird. Events, wie bei den neuen LCS Implementationen gibt es nicht, ist das Ziel in Sicht wird Reward 1.0 weitergegeben.

Single LCS Agent

Mehrere LCS Agenten (Old LCS Agent) teilen sich ein gemeinsames ClassifierSet, das sie entsprechend updaten. Entspricht dem Extremfall der Kommunikation Sight range/Kommunikationsrange

Kapitel 8

Verwendete Hilfsmittel und Software

Zu Beginn stellte sich die Frage, welche Software zu benutzen ist, da es sich um ein recht komplexe Problemstellung handelt. Begonnen habe ich mit der YCS Implementierung von TODO. Sie ist in der Literatur wenig vertreten, die Implementierung bot aber einen guten Einstieg in das Thema, da sie sich auf das Wesentliche beschränkte und keine Optimierungen enthielt.

Der nächste Schritt war zu entscheiden, auf welchem System die Agenten simuliert werden sollen. Unter einer Reihe von vorhandenen Implementierungen entschied ich mich für eine eigene Implementation. Wesentlicher Grund war die Unerfahrenheit mit den Lösungen (und der damit verbundenen Einarbeitungszeit) wie auch Überlegungen bzgl. der Geschwindigkeit, dem Speicherverbrauch und der Kompatibilität. TODO

Das Programm und die zugehörige Oberfläche zum Erstellen von Test-Jobs wurden in Netbeans 6.5 programmiert.

Grafiken wurden mittels GnuPlot erstellt.

Grafiken der Grid-Konfiguration wurden im Programm mittels GifEncode TODO erste * @version 0.90 beta (15-Jul-2000) * @author J. M. G. Elliott (tep@jmge.net)

Wesentlicher Bestandteil der Konfigurationsoberfläche war auch eine Automatisierung

der Erstellung von Konfigurationsdateien, Batchdateien (für ein Einzelsystem und für JoSchKA) zum Testen einer ganzen Reihe von Szenarien und auch GnuPlot Skripts.

Speicherverbrauch

Speicherung der Agentenpositionen und des Grids verbrauchen fast keinen Speicher TODO Wesentlicher Faktor waren die LCS Systeme mit ihren ClassifierSets TODO

Beschreibung des Konfigurationsprogramms

Literaturverzeichnis

[1] Butz, M. & Wilson, S.W.: *An Algorithmic Description of XCS*, 2001. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) Advances in Learning Classifier Systems: IWLCS 2000. Springer, pp253-272.

[2] Larry Bull: *A Simple Accuracy-Based Learning Classifier System*, `http://www2.cmp.uea.ac.uk/~it/ycs/ycs.pdf`

Kapitel 9

Zusammenfassung, Ergebnis und Ausblick

9.1 Zusammenfassung

This is a summary section

9.2 Ergebnis

9.3 Ausblick

THE END...

Anhang A

Statistical significance tests

This is the first appendix

Anhang B

Implementation

The second appendix...

Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabesteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 30. März 2009, Clemens Lode