人 工 知 能 89−14
（1993. 8. 18）

格子空間を移動するエージェント群の協調動作について
−「バベルの塔」における副目標の生成と達成−

山崎哲哉　　渡辺尚

静岡大学工学部

分散協調システムにおいては、最終目標を効率良く達成する副目標の生成が重要な問題となる。本研究では、エージェントの協調動作の基礎的モデルとして提案されているバベルの塔を対象とし、ブロックに関する副目標の生成に焦点を当て、数種のヒューリスティックを開発する。まず、4つの基本アルゴリズムを提案し、副目標を強制的に破棄する機能と、自主的に破棄する機能の2種類の拡張法を示す。アルゴリズムをシミュレーションによって評価し、自主的破棄を伴う遅れ時間最大化アルゴリズムが最も効率が良いこと、および、コストパフォーマンスを最大にする最適エージェント数が存在することを示す。

Coordination of Distributed Agents Moving in a Lattice World

Tetsuya YAMAZAKI　　Takashi WATANABE

Faculty of engineering,, Shizuoka University

3-5-1, Johoku, Hamamatu-shi, Shizuoka, 432 Japan

In order to efficiently achieve a final goal of a basic cordination model, The Tower of Babel, it is important for each agent to creete its successive subgoal. Especially in the model agents have to make crucial decisions on which block they target. This paper exploits four basic heuristics, then shows two additional functions by which an agent may be forced to discard its subgoal and by which an agent discards its subgoal for itself. Simulation results show that Delay Maximizing with No Waiting Time Algorithm is the most efficient algorithm among proposed algorithms. We also find there exists an optimal number of agent which makes the highest cost performance.

1 はじめに

分散協調システムとは、エージェントでは解決できない問題、または非効率的にしか解決できない問題に対して、複数のエージェントが協力・妥協を通して効率的に解を得るシステムである。対象とする問題が大規模であったり、動的な要素を含む場合にはあらかじめ最終目標を達成するための計画を立案することは極めて困難である。分散協調システムでは、最終目標を達成するための最適計画を求めるかわりに、最終目標を効率的に達成し得ると考えられる副目標を各エージェントが自律的に生成する。そして、副目標の生成と達成を繰り返しながら最終目標を達成する。

分散協調システムにおいて、効率良く最終目標を達成するためには、以下の問題を考察しなければならない。

（1） 1つの最終目標に対して、複数の副目標が生成される。つまり、エージェントは大規模な問題を分割しなければならない。問題分割を複数のエージェントで行うか、または単独で行うか、そして、分割をどのように行うかによって、問題解決の効率は、大きく左右される。

（2） あるエージェントにとっての都合の良い副目標が、他のエージェントにとって不都合となることがある。例えば、エージェントが共有し合っている資源に対する競合が発生し、エージェント群が効率良く動作しない場合がある。また、各エージェントが単独で利得の大きい副目標を決定すると、エージェント間で副目標の重複が生じたり、最終目標達成に不可欠な行動をどのエージェントも副目標としない状態も起こり得る。この場合エージェントの副目標の破棄、再生成等の修正が必要となる。また、エージェントが副目標の修正を行なえる時点も問題となる。

（3） 環境の動的変化を考えた場合には、環境の観測、副目標の生成(思考)、副目標の達成(行動)が行なわれることになる。環境の観測、副目標の生成、副目標の達成に必要な時間が、環境の動的変化に対して無視できない場合には、観測と思考と行動のバランスをとらなければならない。例えば、環境を詳細に観測し最適な副目標を生成しようとすると、環境の観測と副目標の生成に要する時間、すなわち、オーバーヘッドが大きくなる。しかも、導出された副目標はもはや最適でないかもしれない。これに対し、副目標を決定できる時点が少ない場合は、各エージェントがとっている行動の一貫性が保持されないため、必ずしも最終目標を効率的に達成できるとは限らない。

エージェント間の協調メカニズムの基礎的考察を行なうためのモデルとしては、Tileworld[1]、追跡問題[2]、バベルの塔[3,4]などが提案されている。Tileworldでは、格子状の2次元平面上にエージェント、タイル、障害物及び穴が存在し、エージェントがタイルを穴に埋めた場合に、得点が得られる。各エージェントの最終目標は、エージェントが、いかに協調して高得点を得るかにある。Tileworldにおいてはエージェント数、穴の出現率、障害物の出現率、穴の得点分布、穴の大きさ分布、穴とタイルとの距離分布、穴の時間的得点変化、穴のタイマ切れ、動作と思考の時間分配、など 非常に多くのパラメータにより、種々の行動選択を検討できる。

また、追跡問題では、格子状の2次元平面上に2種類のエージェントが存在し、逃避エージェントを4つの追跡エージェントが囲むことを目標とする。逃避エージェントの行動を追跡エージェントは予期できない.

一方、バベルの塔は、NTT石田によって自己組織化のための基本モデルとして提案されたものである。このモデルでは、格子状の2次元平面上に1から順に番号のふられたブロックが存在し、エージェントがブロックを中央の台座の上に番号順に載せていく。バベルの塔は、他のモデルに比べて、比較的単純なモデルである。上記（1）の問題については、1つのブロックを台座に運ぶことが分割された問題となる。従って、（1）について考察する余地はあまりない。また、バベルの塔において、環境は、各エージェントの副目標だけに依存して変化する。従って、各エージェントの副目標を瞬時に知ることができると仮定されているバベルの塔では（3）は対象とされない。この意味において、バベルの塔は特に（2）に対して考察を行なうモデルであると考えられる。しかしながら、副目標の生成と達成および、修正の詳細なアルゴリズムについては、十分な知見が得られてはいない。

本研究は、バベルの塔を効率良く解決するヒューリスティックを数種開発し、それらの効率を比較検討することにより、エージェントの副目標生成、修正について考察し、協調メカニズムの基礎を確立することを目的とする。

本稿は、まず、第2章においてバベルの塔を定義し、続いて第3章で、台座に到着した時だけ思考が可能である場合の4つの基本アルゴリズム、さらに、副目標の強制的破棄、自主的破棄を行うアルゴリズムを提案する。第4章では、2エージェントだけが存在する場合を対象として、エージェントが運ぶべきブロックの決定に関するアルゴリズムの評価を行なう。第5章において、エージェント数に対するアルゴリズムの性能を評価する。第6章では、開発したアルゴリズムの中で最も効率の良かったものに対して、格子点における競合がある場合について検討し、第7章にてまとめと今後の課題を述べる。

2 バベルの塔の効率的達成

2.1 バベルの塔

バベルの塔を以下に述べる。

(1) 格子状の2次元平面上に1から順に番号のふられたブロックが散乱している.

(2) 平面上に存在する台座の上に1から順にブロックを積み上げることがバベルの塔の最終目標である.

(3) エージェントは1つのブロックを自分の上に乗せて運ぶ.

(4) エージェントは1単位時間に隣接する格子点に進むことが可能である.

(5) エージェントは問題解決の最中に障害物となるブロックが現れても、それを移動させることはできない.

バベルの塔は、初期状態においてブロックの位置が固定されているため、計画法により最適解が求められる可能性があるが、本研究ではこのアプローチをとらない。それは、以下の理由による。

1） 非常に大きな格子空間や、ブロック数、エージェント数を対象とした場合最適解を求めることは困難である。

2）エージェントが観測可能な範囲を限定すると、複雑な制約付きの問題になってしまう。バベルの塔では全空間を瞬時に見渡せると仮定されるが、現実のロボット制御や、交通システムに適用することを考えると、この仮定はゆるすぎる。

3）バベルの塔の発展形の1つとして、ブロックを動的に発生する環境を想定することは容易に行なえ得る。その場合には、事前に最適計画を立てることが不可能となる。

2.2 バベルの塔における副目標

このバベルの塔においては、以下の2種類の相互に関連しあった副目標が考えられる。この2種類の副目標の決定をどのように行うかにより最終目標の達成の効率は大きく左右されると考えられる。

1）副目標とすべき次のブロックの決定

ブロックを運び終ったエージェントが次に運ぶべきブロックは、エージェントの副目標とみなすことができる。

2）副目標とすべき次の格子点の決定

エージェントが次に移動すべき格子点はエージェントの副目標とみなすことができる.

2.3 バベルの塔における問題点

前章で一般的に述べた、複数のエージェントが協調する場合に生じる問題（2）としては、バベルの塔では、具体的に以下の問題が生じる。

エージェントがブロックに関する副目標を生成する時に、より速く達成できる副目標を生成しようとすると、自分から最も近いブロックを副目標とする。このように、各エージェントが個々により近い副目標の達成を目指した場合、複数のエージェントが同一のブロックを副目標とする副目標の重複が生じる可能性がある。また、台座に載せなければならないブロックをどのエージェントも副目標とせず、最終目標が達成されなくなるという状態も生じる得る。

同様に、格子点に関する副目標を生成する時に、より速く達成できる副目標を生成する。各エージェントが個々に格子点に関する副目標を決定した場合、1つの格子点を複数のエージェントで獲得しようとし、副目標の重複が起こってしまう。このような場合に、エージェントは妥協をする必要がある。

ブロックに関する副目標と、格子点に関する副目標は複雑に関係する。例えば、効率の良いと思われるブロックを副目標としても、格子点に関する副目標決定が適切でない場合には、障害物回避や、他エージェントとの格子点における競合が生じた場合の協調に手間取り、実際には、効率良くブロックを運ぶことができない可能性がある。また、競合している格子点に対し、それぞれのエージェントが副目標としているブロック番号、台座に向かっているかまたはブロックに向かっているかの状態、ブロックや台座との相対位置などの条件を考えて、競合を解消することが効率的な最終目標の達成のためには不可欠である。

さらに、副目標を生成しても障害物や他エージェントの行動によりその副目標を予定通り達成できるとは限らない。また、より効率的な副目標があることが途中でわかることもある。この時、エージェントは現在の副目標を破棄し、新たな副目標を生成する必要がある。副目標が達成できないことや、より効率的な行動があることをエージェントが判断できるとすれば、それは、

（1）エージェントが自らが観測と思考によって知る時

（2）他エージェントからの通信により知る時

の2つの時点に限定される。

3 ブロック決定アルゴリズム

3.1 前提

バベルの塔における2つの副目標は相互に関係している。2つの副目標を考慮した上で、アルゴリズムを開発し、評価した場合、それぞれの副目標決定の長所と短所の考察をすることは困難である。このため、本研究は、副目標とすべきブロックの決定に的を絞って考察する。したがって、格子点における競合はないものとし、1つの格子点は無限個のブロック、エージェントによって共有されると仮定する。また、エージェントは思考を行える時点を限定し、その時点が来た時、副目標を生成し、その副目標の達成を試みる。副目標の生成と達成を繰り返すことによって、バベルの塔の最終目標達成を目指す。

3.2 定義

まず、以下を定義する。

b_j：番号 j を持つブロック

d_j：b_j と台座までの往復の距離

副目標：ブロックの番号とそのブロックを台座まで運ぶステップ数の組 (b_j, d_j)

m：エージェント数

n：ブロックの数

残余ステップ数：エージェントが現在の副目標を達成するまでのステップ数

台座の番号：現在台座の上に載せられているブロックの番号

3.3 基本アルゴリズム

エージェントは副目標を達成した時に次の副目標を決定する。つまり、台座に到着した時に次に運ぶべきブロックを決定し、ブロックを台座まで運ぶために必要なステップ数を予想する。格子点における競合がないため、台座に載せることのできないブロックを台座周辺まで運び、待ち状態になる以外は、予想ステップ数は実際にかかるステップ数と同一となる。

3.3.1 剰余アルゴリズム (R アルゴリズム)

残っているブロックの中で，他のエージェントが副目標としていない最小の番号をもつブロックを選ぶ．つまり，b_{k+m} を目標とする．この場合目標となるブロックは，現在台座に乗せ終ったブロックの番号にエージェント数を加えた番号となる．これを剰余アルゴリズム (Residue Algorithm) と呼ぶ．

3.3.2 待ち時間最小化アルゴリズム (WM アルゴリズム)

台座の番号より2以上大きなブロックを運んだ場合には、自分のブロックが置けるようになるまで台座付近で待たされる。このアルゴリズムでは、他のエージェントが副目標としているブロックよりも大きな番号を持つブロックの中で，待ち時間が最小となるブロックを選ぶ．複数ある場合には，番号が最小のものを選ぶ．ただし、その時点で、他のエージェントが副目標としているブロックの中で最大の番号を持つブロック b_{max} よりも小さい番号のブロックをどのエージェントも副目標としていない場合は、そのブロックを副目標とする。すなわち，以下の条件を満たすブロック b_j を選択し、副目標 (b_j,d_j) を生成する。

$t_l = ssg(x, b_{k+m-1}) + \sum_{i=k+m}^{l-1} d_i - d_l$

x：b_{k+m-1} を副目標としているエージェント

$ssg(c,e)$：エージェント c がブロック e を運ぶための残余ステップ数

$f = min\{u(t_i) \cdot t_i | i = k+m, \cdots, n\}$

u：単位ステップ関数

$V = \{v | t_v = f\}$

$W = \{w | r(b_w) = True, w = k+1, \cdots, k+m-1\}$

$j = minV \cup W$

$r(b)$：他エージェントがブロック b を副目標としていない場合に True を返す述語

エージェントの待ち時間が最小となるブロックを選択しようとするため，これを待ち時間最小化アルゴリズム (Waiting Time Minimizing Algorithm) と呼ぶ．

3.3.3 遅れ時間最小化アルゴリズム (DMin(q) アルゴリズム)

運ぶべきブロックが多い場合には，台座付近で待ち状態にならない限りエージェントは最大限に活動していることになる。しかし，最終ブロックは1エージェントが運ぶため，最終ブロックのスケジュールに失敗すると効率が悪くなる．効率を最大化するためには，最終ブロックの直前のブロックを台座においてから最終ブロックをおくまでの時間をできるだけ短くしなければならない。DMin(q) アルゴリズムでは，残っているブロックの中で番号が最小のものから q 個のうち他のエージェントが台座にブロックを置いてから自分がブロックを置くまでの時間（遅れ時間）が最も小さいものを選ぶ．言い換えると、他のエージェントが b_{k-1} のブロックを運び終った後できるだけ短い時間内に b_k を置くアルゴリズムである．q 個のどのブロックを選んでも待ち時間が生じる場合には、待ち時間が最小となるブロックを選ぶ．ただし、b_{max} よりも小さい番号のブロックで、どのエージェントも副目標としていないブロックが存在する場合は、それらのうち最小のものを副目標とする。このアルゴリズムを遅れ時間最小化アルゴリズム (Delay Minimizing Algorithm) と呼び、以下のアルゴリズムにより、副目標 (b_j,d_j) を生成する。

$if\ (h(T) = True, T = \{t_l | l = k+1, \cdots, k+m-1\})$

$\quad f = min\{t_i | i = k+m, \cdots, k+m-1+q\}$

$\quad V = \{v | t_v = f\}$

$\quad j = minV$

$else$

$\quad f = max\{u(-t_i) \cdot t_i | i = k+m, \cdots, k+m-1+q\}$

$\quad V = \{v | t_v = f\}$

$\quad j = minV$

$h(T)$: 集合 T の全要素が正である場合に True を返す述語

3.3.4 遅れ時間最大化アルゴリズム (DMax(q) アルゴリズム)

DMax(q) アルゴリズムでは DMin(q) アルゴリズムとは逆に、残っているブロックの中で、番号が最小のものから q 個のうち他のエージェントが b_{k-1} のブロックを台座に置いてから自分がブロックを置くまでの時間（遅れ時間）が最も大きいものを選ぶ．これによりあるエージェントが台座に到着してから次のエージェントが到着するまでの時間が大きくなる．また、b_{max} よりも小さい番号のブロックで、どのエージェントも副目標としていないブロックが存在する場合は、それらのうち最小のものを副目標とする。このアルゴリズムを遅れ時間最大化アルゴリズム (Delay Maximizing Algorithm) と呼び、以下により、副目標 (b_j,d_j) を生成する。

$if\ (h(T) = True, T = \{t_l | l = k+1, \cdots, k+m-1\})$

$\quad f = min\{t_i | i = k+m, \cdots, k+m-1+q\}$

$\quad V = \{v | t_v = f\}$

$\quad j = minV$

$else$

$\quad f = min\{u(-t_i) \cdot t_i | i = k+m, \cdots, k+m-1+q\}$

$\quad V = \{v | t_v = f\}$

$\quad j = minV$

3.4 副目標の強制的破棄

最終目標を効率良く達成するために副目標を破棄しなければならない場合がある。副目標を強制的に破棄する方法の1つとしては、他のエージェントが副目標としているブロックを自分が副目標とした方が最終目標の達成には良いと判断した時に、載せ換えを行うことが考えられる。例えば、エージェント A_1 がk番のブロックを台座に載せた場合に、エージェント A_2 が副目標としている b_{k+1} を A_1 が運んだ方が効率が良くなる場合が存在する。このような場合、A_1 からのメッセージによって A_2 が b_{k+1} を中間地点Mまで運び、そして、A_1 が b_{k+1} を台座まで運ぶ。A_2 は、Mで新たに b_{k+2} を副目標とする。この場合、A_2 は、メッセージを受けた時点で、自分の立案したk+1番を台座に載せるという副目標 (b_{k+1},d_{k+1}) を強制的に破棄させられ、k+1番を載せ換え点まで運ぶ新たな副目標 (b_{k+1},p_M) (p_j: 現在の位置から格子点jまでの距離) を生成し、格子点Mでさらに、k+2番を台座に載せる副目標 $(b_{k+2},p_{k+2}+\frac{d_{k+2}}{2})$ を生成することになる。

載せ変えを行った方が効率が良くなる条件を詳細に検討する。載せ変えの対象とするブロックを2個に限定して、今図1のようにエージェント A_1 が b_k を台座に載せた場合を想定する.

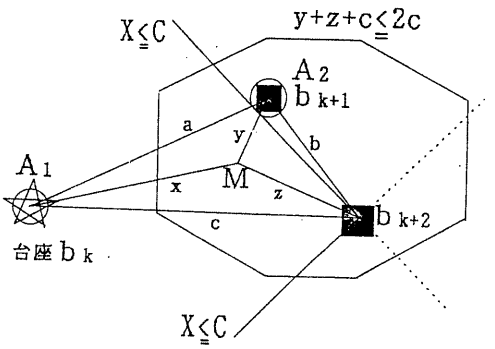

図1：載せ換え点の必要条件

a: b_{k+1} から台座までの距離

b: b_{k+1} から b_{k+2} までの距離

c: b_{k+2} から台座までの距離

x: 台座から載せ換える点までの距離

y: b_{k+1} から載せ換える点までの距離

z: b_{k+2} から載せ換える場所点までの距離

b_{k+1},b_{k+2} を基本アルゴリズムにしたがって運ぶと A_2 が b_{k+1} を台座に置くまでにはaステップかかり、A_1 が b_{k+2} のブロックを台座に置くには $2\times c$ ステップかかる．ここで、大きい値がこの2つのブロックを台座に置くまでのステップ数となる．これを式で表すと以下のようになる．

$S_1 = \max(a, 2\times c)$ (1)

また、b_{k+1} を持ったエージェント A_2 が途中までブロックを運び、そこで副目標を修正して b_{k+2} を取りに行き、置かれた b_{k+1} を A_1 が取りに行く場合には、A_1 が b_{k+1} を台座に置くまでには $2\times x$ ステップかかり、A_2 が b_{k+2} を台座に置くまでには $y+z+c$ ステップかかる．ここで、大きい値の方がこの2つのブロックに関してのステップ数となる．以上を式で表すと以下のようになる．

$S_2 = \max(2\times x, y+z+c)$ (2)

(1) (2) 式より、

$S_1 = 2\times c,\ S_2 = y+z+c$ において $S_1 \geq S_2$ (3)

$S_1 = 2\times c,\ S_2 = 2\times x$ において $S_1 \geq S_2$ (4)

$S_1 = a,\ S_2 = y+z+c$ において $S_1 \geq S_2$ (5)

$S_1 = a,\ S_2 = 2\times x$ において $S_1 \geq S_2$ (6)

の4つの場合には、載せ換えを行った方が効率が良くなる可能性がある。(4),(6) 式が成立する状況では効率は良くなる可能性はあるが、$2\times x \geq y+z+c$ であるため、載せ換えを行うと、k+1番のブロックよりもk+2のブロックを先に持ってきてしまうため、A_2 が待ち状態になってしまう。

これに対し、(3),(5) 式は、載せ換えを行うと効率が向上すると考えられる。このため、(3),(5) 式の条件に当てはまった場合に載せ換えを行うこととし、この場合に、台座にいるエージェントが載せ換えを行うエージェントにメッセージを送り、メッセージを受信したエージェントは副目標を修正し、指示された格子点にブロックを置き、次のブロックを取りに行くとする。

(3) 式の載せ換え点の範囲を考察する。まず、$2\times c \geq y+z+c$ の関係が成り立つためには $c \geq y+z$ となる範囲を図示すると、b_{k+1} と b_{k+2} からの距離の和がc以下の範囲、つまり、b_{k+1} と b_{k+2} を焦点とする楕円の内側となる．さらに、$y+z+c$ の最大値は $2\times c$ であるため、$2\times x \leq 2\times c$ つまり $x \leq c$ が成り立たなくてはならない．この範囲を図示すると、台座からの距離がc以下の範囲、つまり台座を中心とする半径cの円の内側となる．この2つの条件を共に満たす領域に置くことが必要条件となる．

また、この領域内のどの格子点を載せ変え点とするかが問題となる。考えられるのは、k+1番のブロックをできるだけ速く台座に載せられる格子点、または、k+2番のブロックをできるだけ速く台座に載せられる格子点である。

3.5 副目標の自主的破棄

現在の副目標を達成できないとわかった時、自主的に副目標を破棄し、次の副目標を決定する。つまり、台座の番号がk番である時、k+2番以上のブロックを持ったエージェントが台座に接近すると、エージェントはブロックを台座に載せられずに、待ち状態となる。エージェントが台座の近傍で待ち状態となることを判断した場合には、その場にブロックをおいて、次の副目標を決定するものとする。

4 各アルゴリズムの評価

格子点における競合がない場合において、考案したブロック獲得アルゴリズムをシミュレーションによって評価する．ここで以下の用語を定義する。

下限LB： 全てのブロックの台座からの往復の距離の和をエージェント数で割ったもの．下限は完全に全てのエージェントが並行に動作し、無駄なく全てのブロックを運んだ場合のステップ数である。このためシミュレーションにおいてステップ数が下限よりも小さくなることはない．

$$\mathrm{LB} = (\text{総距離})/(\text{エージェント数}) = \frac{\sum_{j=1}^{n} d_j}{m} \quad (7)$$

平均増加率AD： 各サンプルにおけるステップ数の総和を下限の総和で割ったもの．

$$\mathrm{AD} = \frac{\sum_{k=1}^{n} s_k}{\sum_{k=1}^{n} LB_k} \quad (8)$$

s_k：サンプルkにおけるステップ数

LB_k：サンプルkにおける下限

平均増加率＝1は理想値である．

4.1 シミュレーション仮定および結果

本研究では、図2に示す仮定において、シミュレーションを行った．

空間の大きさ	200x200
エージェント数	2
ブロック数	1000
台座の位置	格子空間中央
エージェントの初期位置	台座
ブロックの初期位置	ランダム
サンプル数	100

図2 シミュレーション仮定

4.2 基本アルゴリズムの評価

シミュレーションの結果に基づいて、基本アルゴリズムの評価を行なった。シミュレーション結果を図3に示す。

基本アルゴリズム	平均増加率	平均ステップ数
R	1．103	110864
WM	1．058	106304
DMin（5）	1．059	106347
DMax（5）	1．049	105435

図3　基本アルゴリズムの評価

4.2.1 Rアルゴリズム

このアルゴリズムはエージェントが台座に到着して、次に副目標とするブロックを決定する際に、他のエージェントとの通信の必要もなく非常に容易に次に副目標とするブロックが決定できる。しかし、効率を考えないため、台座の番号がk番である場合にb_{k+2}を持ったエージェントがb_{k+1}を持ったエージェントよりも速く台座に到着し、エージェントがブロックを台座に載せられず待ち状態になってしまう場合が多い．このため、図3からもわかるように効率は良くない。

4.2.2 WMアルゴリズム

図3より、このアルゴリズムは、Rアルゴリズムよりも効率が良くなっていることがわかる．このことから、待ち状態を少なくすることが効率向上につながると言える。しかし、このアルゴリズムは待ち時間が0となるブロックが存在しない場合には全部のブロックを探索してしまう。また、副目標としたブロックを置き終った時の状態を考慮していないため、このアルゴリズムによって選んだブロックを副目標としたために後で待ち状態に陥ってしまう可能性がある。このため待ち時間が0となるブロックが複数存在した場合番号が最小のものを選ぶことが必ずしも最適であるとはいえない。

4.2.3 DMin(q)アルゴリズム

シミュレーション結果はWMよりも悪くなった。これは以下のように考察できる．

このアルゴリズムでは、2つのエージェントがほぼ同時に台座に到着した場合、先に到着したエージェントは相手エージェントの残余ステップ数が小さいために待ち時間が0となるブロックが複数存在する．これに対し、後に台座に到着したエージェントは先に到着したエージェントが目標のブロックに対する行動を開始して時間が経っていないため、相手エージェントの残余ステップ数が大きくなり、待ち時間が0となるブロックが存在せず、どのブロックを副目標としても待ち状態となってしまう確率が高くなる．先に台座に到着したエージェントにとって待ち時間が0となるブロックが多く存在しても結局副目標とするブロックは1つであるため、後に台座に到着するエージェントの悪影響によって結果として効率は悪くなってしまう．このため、ブロック数が多い場合最終ブロックで効率が良くなっても途中の非効率性が影響して全体として効率が悪くなってしまうと考えられる．このため、効率の良いアルゴリズムとは言えない．

また、qに関しては、q=1からq=20までシミュレーションを行った。図4に示すように、その結果、q=4以上ではほとんど同じ結果となった。このため、格子空間が200×200の場合はqは4ないしは5で良く、それ以上

先のブロックを見てもあまり効率が変わらないと言える.

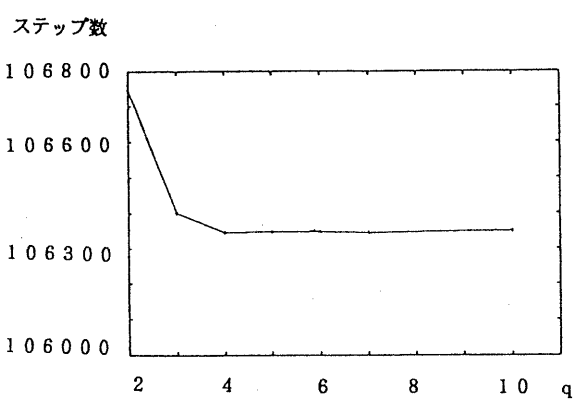

図4 : DMin(q) における q に対する評価

4.2.4 DMax(q) アルゴリズム

このアルゴリズムは DMin(q) とは逆にエージェントが台座に到着する時間をできるだけ離すため, エージェントが台座に到着した時に、相手エージェントの残余ステップ数は DMin(q) のそれほど小さくはなく、待ちが 0 となるブロックの数は減少する. DMin(q) の後で到着するエージェントの計算による相手エージェントの残余ステップ数ほどここでの相手エージェントの残余ステップ数は大きくならない. このため、エージェントが台座に到着して次のブロックを決定する場合に、待ちが 0 となるブロックが存在する確率が常に高く、それによって待ち時間が減少し、効率が上がると思われる. ただし、残りのブロックが少なくなり、m 以下になった時には、効率は低下してしまう。

図3からすれば、DMax(q) は、基本アルゴリズムの中で最も効率の良いアルゴリズムである。

q に関しては DMin(q) と同様に q=4 で十分であるとの結果が得られた。

4.3 副目標の強制的破棄を含むアルゴリズムとその評価

基本アルゴリズムに副目標の強制的破棄を加えたアルゴリズムを開発した。そのアルゴリズムを説明するとともにシミュレーションによる評価を行なう。シミュレーション結果を図5に示す。

また、以下のシミュレーションでは DMax(5) を基本アルゴリズムとして採用している。

	アルゴリズム	平均増加率	平均ステップ数
副目標の強制的破棄	PDM	1.051	105629
	PSM	1.050	105632

図5 : 強制的破棄を付加したアルゴリズムの評価

4.3.1 PDM アルゴリズム

載せ換えをする条件を満たし、格子点を定める場合には b_{k+1} を最も速く置くことができる点、つまり図1において、x が最小となる格子点を載せ換え点として選択する。これを部分遅れ時間最大化アルゴリズム (Partial Delay Maximizing Algorithm) と呼ぶ.

このアルゴリズムでは、目標の修正により、b_{k+1} のブロックを置く時間は遅れ、b_{k+2} のブロックは速く台座に置くことができるため、2つのエージェントの台座への到着時間が DMax(q) よりも近くなる. したがって DMin(q) アルゴリズムで行った考察と同様に、b_{k+2} を持ったエージェントの相手エージェントの台座間距離が大きくなり、どのブロックを担当しても待ちになるという非効率的な状態になる. また、載せ換え点に向かう場合に b_{k+1} のブロックにとっては遠回りして置かれることになり b_{k+3} を取りに行く時刻が目標修正をしない場合に比べて遅くなってしまうことがある. このため、目標修正による部分的効率向上に対して非効率的な状態が与える影響が大きく、全体として効率は DMax(q) と比べてあまり変わらなかったと考えられる。

4.3.2 PSM アルゴリズム

載せ換えをする条件を満たし、格子点を定める場合には b_{k+2} を最も速く台座に載せることができる点、つまり図1において、y+z+c が最小となる格子点を載せ換え点として選択する. これを部分ステップ最小化アルゴリズム (Partial Step Minimizing Algorithm) と呼ぶ.

このアルゴリズムでは、載せ換え点に向かう場合に b_{k+1} のブロックにとっては遠回りすることになり b_{k+3} を取りに行く時刻が遅くなってしまうことがある. b_{k+2} をできるだけ速く台座に載せることのできる格子点を載せ換え点とするため、この影響は PDM で受けるものよりも大きい影響を受ける. さらに、エージェントが台座に到着する時間が近づいてしまうことから DMin のように後に台座に到着するエージェントがどのブロックを副目標としても待ちの状態になってしまうことによる影響も考えられる.

またこのアルゴリズムでは $2 \times x = y + z + c$ となる格子点でブロックの受け渡しをすることがある. この場合エージェントは同時に台座にブロックを載せようとするが、台座には同時にブロックを置くことができないため、番号の大きいブロックを持ったエージェントが一端待つ状態になってしまう. シミュレーション2では、ブロックが 1000 個であるため、もし $2 \times x = y + z + c$ の状態が全てのブロックに対して起こった場合 2000 ステップ余分にかかってしまう. このような影響により、このアルゴリズムの効率は DMax(q) と比べてあまり変わらなかったと考えられる。

以上をまとめると、PDM,PSM は DMax とほぼ等しいかまたは若干性能が悪い。本研究で提案した副目標の強制的破棄は、2つのブロックしか対象としていないため、3つ以上先のブロックがうまく選択されていないことが原因であると考えられる。

4.4 副目標の自主的破棄を含むアルゴリズムとその評価

基本アルゴリズム、副目標の強制的破棄に副目標の自主的破棄を加えたアルゴリズムを評価した。シミュレーションによる評価結果を図６に示す。

		アルゴリズム	平均増加率	平均ステップ数
副目標の破棄	自主的破棄	DMax-NW	１．００１	100605
副目標の破棄	強制的破棄	PDM-NW	１．００１	100609
副目標の破棄	自主的破棄	PSM-NW	１．００２	100687

図６：強制的破棄と自主的破棄を付加したアルゴリズムの評価

4.4.1 DMax-NW アルゴリズム

基本アルゴリズムで最も効率の良かったDMaxアルゴリズムに加え、エージェントが待ち状態になった場合に副目標の自主的破棄を行なう。これをDelay Maximizing with No Waiting Time Algorithm(DMax-NW アルゴリズム)と呼ぶ。

このアルゴリズムは、平均増加率が１に近く、ほとんどステップ数の下限に近い値が得られた。これは、以下のように考察できる。

このアルゴリズムにおいては、効率を悪くする要因としては、以下の２点だけが考えられる。

1） b_{n-m+1} から b_n までのブロックの積み方が非効率的になる。

2） 台座付近にブロックを置いて次のブロックをとりにいく場合に、置く格子点に対して次のブロックが台座をはさんだ方向にある場合には２ステップだけ効率が悪くなる。

待ち状態になる確率が約２０%、その時次に副目標とすべきブロックが台座をはさんだ方向にある確率は５０%であるため、第２の要因に関する影響は、ブロック数１０００においては高々２００である。また、第１の要因に関してはmが小さい範囲では、効率への影響はあまり大きくない。従って、無駄な行動が少ないアルゴリズムであると考えられる。

4.4.2 PDM-NW アルゴリズム

PDM(q) アルゴリズムに目標の自主的破棄を加える。これをPartial Delay Maximizing with No Waiting Time Algorithm(PDM-NW アルゴリズム) と呼ぶ.

このアルゴリズムは、PDMアルゴリズムでの考察同様、b_{k+1} を速く台座に載せることを考えているため、b_{k+3} を取りに行く時刻が遅くなってしまう影響があるため、DMax-NW アルゴリズムよりもやや効率が悪くなってしまったと考えられる。しかし、各サンプルの結果を見てみると、DMax-NW アルゴリズムよりも良い結果が得られているものがあった。b_{n-m+1} のブロックがおかれてから b_n をおくまでのステップ数を見ると、DMax-NW とほぼ同じであるのに、全体のステップ数はこのPDM-NW の方が良いという結果になっているものがあった。このため、このアルゴリズムにおいて、b_{k+3} を取りに行く時刻が遅くなってしまう場合には、目標の強制的破棄を行なわないようにすると、効率を改善できる可能性がある。

4.4.3 PSM-NW アルゴリズム

PSM(q) アルゴリズムに目標の自主的破棄を加える。これをPartial Step Minimizing with No Waiting Time Algorithm(PSM-NW アルゴリズム) と呼ぶ.

このアルゴリズムでは、b_{k+2} をできるだけ速く台座に載せることのできる格子点を載せ換え点とするため、b_{k+3} を取りに行く時刻が遅くなる影響が大きい. この影響により、DMax-NW、PDM-NW に比べて効率は悪くなったと考えられる. シミュレーション結果を見ても、DMax-NW、PDM-NW に比べて、どのサンプルにおいても効率が悪くなっている。このため、このアルゴリズムは、効率が悪いといえる。

4.5 アルゴリズムの総合評価

シミュレーションの結果に基づき効率の良かったものを以下にまとめる。

基本アルゴリズム
　DMax(q) アルゴリズム

基本アルゴリズム＋副目標の強制的破棄
　PSM アルゴリズム

基本アルゴリズム＋副目標の自主的破棄
　DMax-NW アルゴリズム

そして、全てのアルゴリズムの中でDMax-NW アルゴリズムが最も効率が良いという結論が得られた。

5 エージェント数に対する評価

開発したアルゴリズムの中で最も効率の良かったDMax-NW アルゴリズムについてエージェント数の影響を考察する。

このシミュレーションの評価には、以下の式で表されるコストパフォーマンス pc を用いる。

$$pc = \frac{S_1/S_m}{S_m * c_1 * m + c_2 * m} \tag{9}$$

S_m ： mエージェントで解決させた場合にかかるステップ数

c_1 ： １ステップにかかるコスト

c_2 ： １エージェントにかかるコスト

分子 S_1/S_m は1エージェントで目標を達成する場合と比べてmエージェントで行なった場合、何倍の速度で目標を達成できたかを示す性能を表す。また、分母第1項はエージェントのステップ数に比例してかかるランニングコストを表し、第2項はエージェント数に比例する初期コストを表している。

シミュレーション結果を図7に示す。

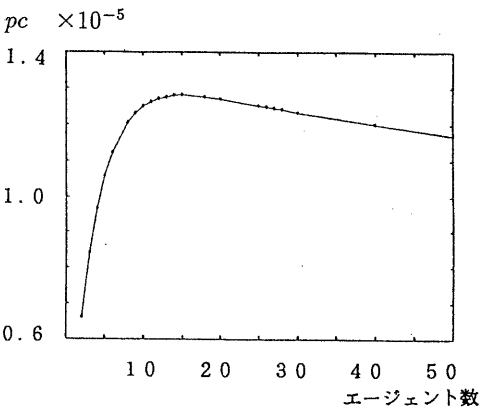

図7：エージェント数に対するDMax-NWの性能

c_1：c_2＝1：50000とした場合、図7より、エージェント15が最もコストパフォーマンスが高く、性能とコストのバランスが最も良かったことがわかる。

エージェントが2から14では、エージェントにかかるコストの上昇よりもステップ数の減少による性能の上昇が上回り、全体として pc は上昇していく。エージェントが16以上では、目標達成のステップ数が減少し、性能は上昇する。しかし、エージェント数が14以下の場合と比べ、その上昇率は低く、性能は飽和する。これに対し、分母は、エージェント数に対して増加する。このため全体として、pc は下降する。c_1 と c_2 の比によって pc の最適点は変化するが、最適点の存在は同様に説明できる。

6 格子点における競合がある場合

バベルの塔において格子点における競合がある場合を検討する。エージェントは進路方向に目的としないブロックが存在した場合、これを回避しなければならない。また、他エージェントと格子点を共有することができないため、複数のエージェントが同一の格子点に移動しようとした場合には、エージェントが妥協をして他の格子点に移動するか、または、その格子点に留まらなければならない。本研究では、このように格子点競合が生じた場合には、より小さい番号を副目標としているエージェントを優先させるものと仮定して、シミュレーションを行なった。ただし、エージェントの初期位置は台座ではなく、ランダムに決定する。また、副目標が永久に達成できない場合は、失敗とみなす。

このような仮定でDMax-NW,PDM,PSMアルゴリズムを用いたシミュレーションを行なった。結果を図8に示す。

エージェント数	成功サンプル数	平均増加率	平均ステップ数
2	62	1.022	102665
3	29	1.037	69331
4	0	-------	--------

図8：格子点競合がある場合のDMax-NWの評価

DMax-NWでは、格子点における競合がない時と比べて、エージェント数が2の場合は、約2000ステップ効率が悪くなり、エージェント数が3の場合は、約1700ステップ効率が悪化した。初期位置の違いはあるものの、ブロック数が1000個であるため、1つのブロックを運ぶために約2ステップ余分にかかっていることがわかる。

また、格子点における競合があるため、最終目標を達成できなくなる状態の頻度が問題となる。エージェント数2では100のサンプルに対して成功したサンプルは62、エージェント3では29であり、4以上のエージェント数では、どのサンプルに対しても目標を達成することができなかった。

DMax-NWアルゴリズムでは、台座の番号がk番であるのに対し、k+2番以上のブロックを持って来た場合、待ち状態になることを判断し、その格子点にブロックをおいて次のブロックをとりに行く。この待ち状態を判断できるのは、台座の上下左右の4つの格子点であると限定している。この4つの格子点全てに、k+2番以上の番号のブロックをおいてしまった場合、k+1番のブロックを持ったエージェントが、台座に到着できずにデッドロックとなってしまう。エージェント数が多くなるほどこの状態になる可能性は高い。

また、格子点における副目標決定アルゴリズムが単純であるため、エージェントは障害物となるブロックが複雑に存在すると、それを回避できずに目標を達成できなくなるという状態も起こる。このためDMax-NWアルゴリズムにおいて、待ち状態になった場合にブロックを置く格子点の決定アルゴリズムを開発しなければならない。この解決策の1つとしては、台座上のブロックの番号と自分が持っているブロックの差に比例した距離の地点に置く方式が考えられる。

また、PDMアルゴリズムにおいて格子点競合がある場合のシミュレーションを行なった。シミュレーションの結果、格子点競合がない場合と比較して、約3000ステップ効率が悪化していた。これは、以下のように考えられる。副目標を強制的に破棄させた場合に、どちらのエージェントも障害物により、副目標通り載せ換え点に到着できるとは限らない。また、メッセージを受けたエージェント A_2 が載せ換え点Mに予定通り到着できない場合は、メッセージを送ったエージェント A_1 が載せ換え点で待ち状態となってしまい、効率が悪化する。また、メッセージを送ったエージェント A_1 が副目標 (b_{k+1},p_M) より遅れた場合は、A_2 が b_{k+2} を b_{k+1} よりも先に台座付近に持って来てしま

い、待ち状態になってしまう可能性がある。このため、格子点競合がない場合と比較して、効率が悪化したと考えられる。

さらに、PSM アルゴリズムにおいて格子点競合がある場合のシミュレーションを行なった。シミュレーションの結果、格子点競合がない場合と比較して、約１００００ステップ効率が悪化していた。これは、PDM 同様、メッセージを受けたエージェント A_2 が副目標 (b_{k+1}, p_M) 通り載せ換え点に到着しない場合にも効率は悪化する。また、PSM アルゴリズムでは、副目標を強制的に破棄させて、b_{k+1}、b_{k+2} を運んだ場、b_{k+1} と b_{k+2} はほぼ同時に台座に置かれることになる。このため、メッセージを送ったエージェント A_1 が少しでも予定通り台座に到着できない場合、メッセージを受けたエージェント A_2 は、b_{k+2} を b_{k+1} よりも先に台座に持って来てしまい、台座付近で待ち状態となる確率が高い。このため、PDM に比べてもさらに効率が悪化すると考えられる。

以上より、副目標の強制的破棄を行なう場合は、エージェトが予定通り載せ換え点に到着する、また、台座に到着するとは限らないため、効率の悪化が大きいと考えられる。エージェントが予定通り載せ換え点に到着できない場合には、相手エージェントとの通信により、載せ換え点を更新することが今後の拡張として考えられる。

7 まとめ

本研究はエージェントの基礎的協調動作に関して提案されているバベルの塔に関して問題となる

（１）副目標とすべき次のブロックの決定

（２）副目標とすべき次の格子点の決定

において、（１）の問題に関して効率の良いヒューリスティックの開発を目的とした。開発したアルゴリズムとしては、副目標を達成した時に次の副目標を決定できる４つの基本アルゴリズムを示し、さらに、１つのブロックを２つのエージェントで運んだ方が効率が上がると判断した場合に副目標を破棄させる副目標の強制的破棄と、エージェントが持ち状態になった場合に副目標を破棄する副目標の自主的破棄の機能を付加した５つのアルゴリズムを開発した。シミュレーションの結果、２エージェントでは開発したアルゴリズムの中では、DMax-NW アルゴリズムが最も効率が良いという結論が得られた。この DMax-NW アルゴリズムに関してエージェント数に対する評価を行ったところ、コストパフォーマンスを最大にするエージェント数が存在することがわかった。しかし、この DMax-NW アルゴリズムでは、格子点競合がある場合には、大きな番号のブロックが台座を囲んでしまい、最終目標が達成できなくなってしまう。

また、本研究で行なった基本アルゴリズムの２種類の拡張は別の言い方をすれば、副目標修正の機会を多くすることによる拡張であるといえる。機会を多くすることが、必ずしも効率を上げることには結び付かないことがシミュレーション結果から考察できる。したがって、基本アルゴリズムに副目標の強制的破棄を付加した DMax-NW を基にして、格子点獲得に関する効率の良い副目標決定アルゴリズムの開発およびブロックの置き場所の検討が今後の課題となる。なお, 本研究は文部省科学研究費 (課題番号 0520820 の助成を受けて行なった.

[参考文献]

[1]Martha Pollack and Marc Ringuette: "Introducing the Tileworld: Experimentally Evaluating Agent Architectures." In Proceedings of The Eighth National Conference on Artificial Intelligence, pp.183-189,1990.

[2]M.Benda,V.Jagannathan,and R.Dodhiawalla." On optinal cooperation of knowledge sources." Proceedings of the 1988 Workshop on Distributed Artificial Intelligence, May 1988.

[3] 石田亨："バベルの塔：組織指向プラニングに向けて Tower of Babel: Towards Organization-Centered Planning", 第一回マルチ・エージェントと協調計算ワークショップ (MACC '91) 資料 (1991)"

[4]Toru Ishida:" Towards Organizational Problem Solving" , IEEE CONF. ON Robotics and Automation 1993