

artisoc で作成する初等情報教育用 OSPF ネットワークルーティングシミュレーション

末 木 俊 之

OSPF Network Routing Simulation with Artisoc for Beginning Class Information Education

Toshiyuki SUEKI

キーワード：自律システム（AS） リンク状態データベース 最短経路ツリー 経路制御表 ダイクストラ（Dijkstra）のアルゴリズム

1. OSPF ネットワークルーティングとシミュレーション概要

昨年度の研究紀要¹⁾では、マルチエージェントシミュレーションソフト artisoc textbook²⁾を使い電子回路シミュレーションを作成した。これは、情報処理科目での使用を意図したものである。

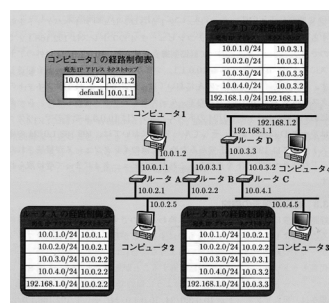
今回も同様に、ネットワーク（インターネット）の仕組みについて講義する折に使用するツールを目指し、artisoc textbook を使ったシミュレーションを作成した。

パケットを宛先のコンピュータに届ける経路制御は、パケットを受け取ったコンピュータやルータでの転送先決定判断に負っている。パケットはパケットリレー方式で次の転送先に送られる。個々のコンピュータ・ルータは経路制御表を保持しており、受け取ったパケットの宛先 IP アドレスに基づいて、経路制御表を検索し次の転送先を決定する。

経路制御表とそれに基づいたパケット転送は、(図1.)によって説明できる。

経路制御表は、宛先 IP アドレス（宛先）とネクストホップ（次の転送先）の2列の表で表

せる。宛先 IP アドレスに該当する行を見つければ、同行のネクストホップ列に載っているアドレスが転送先として決定される。



(図1.) 経路制御表について：(『Computer Science Library 8 コンピュータネットワーク入門 - TCP/IP プロトコル群とセキュリティ-』より³⁾)

パケットの宛先が、ルータが直接接続するネットワーク（データリンク）上のコンピュータである場合には、ネクストホップはルータの持つ宛先ネットワーク側の IP アドレスとして記述されている。他のネクストホップはルータの IP アドレスになっている。基本的には、パケットは経路制御表に基づいて次に転送すべきルータが決められ送信されると説明できるで

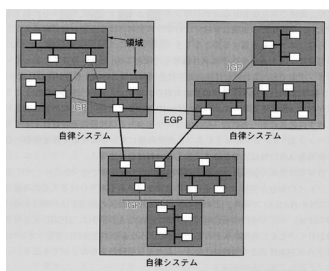
あろう。

ただし、これで経路制御についての説明を終わらせてしまうと、「では、経路制御表とはどのように構築されるのだろうか？」という疑問が残る。この辺りに少しでも触れないとインターネットの経路制御について理解できたという感覚を抱きにくい。

インターネット全体について経路制御を詳細に説明するというのは骨が折れるが、自律システム（AS: Autonomous System）内における経路制御について説明する場面では、適当なシミュレーションがあれば可能であろう。

自律システム（AS: Autonomous System）は、経路制御ドメインとも呼ばれ、地域のネットワーク、規模の大きな ISP（Internet Service provider）など、組織内で経路制御に関する方針である運用ポリシーを決め、その運用ポリシーを基に経路制御を運用する範囲とされている。

経路制御プロトコルは、IGP（Interior Gateway Protocol）と EGP（Exterior Gateway Protocol）の2つに分類される。（図2.）で説明されるように IGP は自律システム内の経路制御を行うプロトコル、EGP は自律システム間の経路制御を行うプロトコルである。



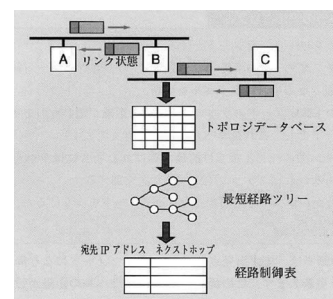
（図2.）IGP と EGP について：（『Computer Science Library 8 コンピュータネットワーク入門 - TCP/IP プロトコル群とセキュリティ-』より⁴⁾）

IGP の代表的なプロトコルとしては、OSPF（Open Shortest Path First）や RIP（Routing Information Protocol）がある。今回取り上げたのは OSPF を説明するためのツールである。

OSPF ルーティングでは、各ルータは自分が直接隣接しているルータの情報をパケットで知らせ合い、AS 内全てのコンピュータ・ルータが自分の内部にリンク状態のデータベースを保持する。OSPF ルーティングに参加するコンピュータ・ルータが AS 内全ルータに関する同一のリンク状態データベースを持つ。そして、各コンピュータ・ルータはデータベースを探索し自分を起点とするネットワーク上の他のノードへの最短経路ツリーを計算する。最後に最短経路ツリーから経路制御表を作成する。

（図3.）で図示される①～③の手順で経路制御表データが作成される。

- ①リンク状態データベース（トポロジーデータベース）を作る。
- ②自分が起点（根）となる最短経路ツリーを作る。
- ③最短経路ツリーから経路制御表を作成する。



（図3.）OSPF ルーティング：（『Computer Science Library 8 コンピュータネットワーク入門 - TCP/IP プロトコル群とセキュリティ-』より⁵⁾）

- ①リンク状態データベース（トポロジーデータベース）を作ること。

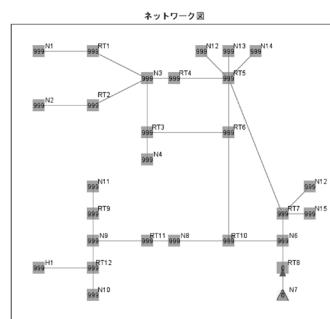
簡単に言ってしまうと、リンク状態データベースとは、「あるノード（A）が、どのノード（B）に隣接してつながっているか。またA→Bへパケット転送する場合のコストはいくらか」というノードの連結関係のデータベースであろう。昨年度作成した artisoc による電子回路シミュレーションでは、link エージェントを生成させて連結関係データを保持させたが、この link エージェント全体を、リンク状態データベースとして使える。電子回路シミュレーションで使った link エージェントに、連結のコスト（メトリックとも呼ばれる）値を追加すればよい。artisoc を使えばルータ・ネットワーク等をノードとして四角形の画像で表示させ、連結するノード同士を線でつないで表示するのは容易である。

ルータ、ネットワーク等のノードを画面上に表示させるには、電子回路シミュレーションと同様に、表示位置座標等の情報を保持する node エージェントを使う。link エージェント、node エージェントの初期値データはエクセルシート上で作成し、CSV テキストファイルで保存し artisoc に読み込ませる。そして、シミュレーションの動作直後に全 link エージェントデータを基に、node エージェント同士を連結する処理を行う。この辺りも、昨年度作成した artisoc による電子回路シミュレーションと同様である。

例えば（図 4.）のようなルータ等のつながりを表すネットワーク図を artisoc で表示するのは、電子回路シミュレーションと同様に容易である。この図は、リンク状態データベースを視覚的に表現したものと言えるであろう。（図 4.）は、参考にした書籍で OSPF ルーティング解説に使われているネットワーク例⁶⁾を参考に artisoc で再現したものである。白黒印刷では分かりにくいですが、実際は隣接したノード同

士は青色の実線でつながれている。この例図程度のネットワークでも最終的に経路制御表を実際に手作業で作る演習を実行するのはかなり困難である。シミュレーションが使って最短経路ツリー作成が簡単にできる環境がなければ難しい。今回のシミュレーション作成を思い立ったきっかけとなった例図である。

ノードがネットワーク（データリンク）であるものは、ノードの近辺に“N”で始まる名前が表示されている。N1～N15はネットワークである。またノードの近辺に“RT”で始まる名称が表示されているノードはルータである。RT1～RT12はルータである。各ノードが保持するリンク状態データベースは、実際はデータの塊であるが、artisoc で図示してみると視覚的に全体構造が捉えられるので分かりやすい。



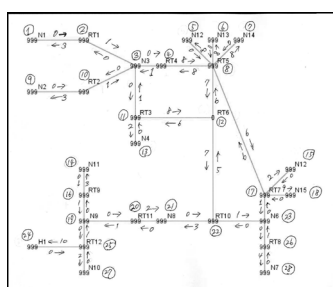
（図 4.）ネットワーク図

（図 4.）のネットワーク図で、経路制御的にはルータ同士が直接連結した辺がない違和感がある部分もある。例えばルータ RT1、RT2、RT3、RT4は全てネットワーク N3につながっているのですが、実際の経路制御ではそれらのルータ同士が直接に連結される線があるべきではないかとも思われる。参考にした書籍によれば、実際のリンク状態データベースでは、（図 4.）のようなネットワーク図に相当する連結関係を表現した情報となっている。

また、実際の詳細 OSPF ルーティングにお

けるリンク状態データベースの内容とは若干の相違があるが、今回のシミュレーションは（図5.）で示したコスト値を使っている。（図5.）におけるノード同士をつなぎ辺に2つずつ手書きした矢印・数値は、辺上で矢印方向にパケットが通過するときのコスト値を表現している。

手書きの丸数字は、各ノードを識別するための番号である。このシミュレーションには全部で28個のノードが存在するので、各ノードには1から28の一意の数字が付加されている。



（図5.）ノードIDno・コスト値を手書きで付加したネットワーク図

②各ノードが自分を起点（根）とする最短経路ツリーを作ること。

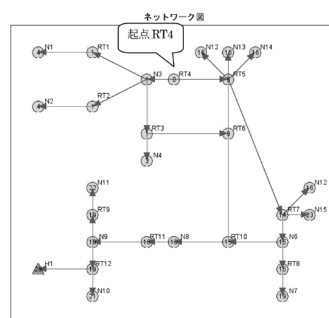
ネットワーク上の各ノードは、自分を起点としたAS内全ノードへの最短経路ツリーを作る。

最短経路探索には、ダイクストラ（Dijkstra）のアルゴリズムが使われる。

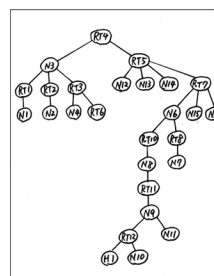
（図6.）は、RT4ルータを起点する最短経路ツリーを artisoc で描いた例である。白黒印刷では分かりにくいですが、実際はRT4を起点として赤い矢印で全ノードが連結している。また、各ノードには、起点から自分に至るまでのコスト合計値を表示している。

（図6.）で最短経路ツリー図が完成したとしてもよいのであろうが、（図6.）を元に、情報理論データ構造の説明で登場するようなツリー図を手書きで描くならば、（図7.）のような図が描ける。

また artisoc は、シミュレーションの条件（初期条件等）をコントロールパネルによって変え



（図6.）RT4ルータを起点とした最短経路ツリー



（図7.）手書きの最短経路ツリー

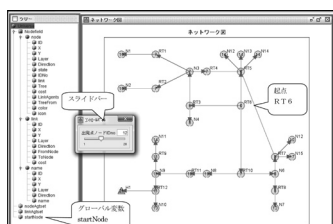
ることができるので便利である。（図8.）のように、スライダーを動かすことにより出発点ノードを変えられることが可能にしてみた。スライダーの操作と、（図8.）左のツリー最下部に表示されるグローバル変数 startNode が連動し、変数値を1～28の間で変えられる。node エージェントには、ノードを識別するための整数型変数 IDno を持たせてある。この例では、ノード数は28個で、各ノードには1～28の一意の値がふられている。そして、startNode 値と同じ IDno を持つノードを起点にしてシミュレーションが動作するようにした。

（図8.）の例では、スライダーの示す値が12なので、IDno =12を持つRT6ノードを起点とした最短経路ツリーが作成される。

スライダーを動かすことにより、画面上の

全てのノードを起点とする最短経路ツリーを作成させることができる。

③最短経路ツリーから経路制御表を作成すること。



(図8.) スライダーの操作により、起点をRT6にしたシミュレーション

artisoc シミュレーションで描かせた最短経路ツリー図 (図6.) または (図7.) から経路制御表を作成するのは簡単である。宛先 IP アドレス、ネクストホップの IP アドレスの代わりに、単にノードの名称を使うとすれば、(表1.) のような経路制御表が作成できる。

(表1.) でネクストホップに黒星印 (★) を付加した個所は OSPF ルーティングの実情と

宛先	ネクストホップ		宛先	ネクストホップ		宛先	ネクストホップ
N3	N3		N14	RT5		N9	RT5
RT1	N3	★	RT7	RT5		RT12	RT5
RT2	N3	★	N12	RT5		H1	RT5
N1	N3	★	N15	RT5		N10	RT5
N2	N3	★	N6	RT5		RT9	RT5
N4	N3	★	RT10	RT5		N11	RT5
RT6	N3	★	N8	RT5		RT8	RT5
RT5	RT5		RT11	RT5		N7	RT5
N12	RT5		N9	RT5			
N13	RT5		RT11	RT5			

(表1.) ルータ RT4における経路制御表の例

は合っていない箇所であろう。パケットはネットワーク N3内を通過するので、N3への経路のコストも加算されるという考えには違和感がないが、この場合のネクストホップはルータでなければならない。実際には N3では無く、適切なルータがネクストホップに選ばれるはずである。この部分は初等情報教育での経路制御の概要説明ということで簡略化して扱うことにした。

2. OSPF アルゴリズムシミュレーションのエージェント

(1) node エージェント

エージェントを定義すると、自動的に ID、X、Y、Direction、icon などの変数が用意される。特にシミュレーションでは使用していない変数は、(未使用) と記述する。

変数値はエクセルシート上で作成し、CSV テキストファイル化したものを初期値として読み込ませている。

変数名	型	用途
ID	整数型	artisoc エンジンが自動的に各エージェント識別用に設定する値(未使用)
X	実数型	エージェントの X 座標値
Y	実数型	エージェントの Y 座標値
Layer	整数型	ノードを表示するレイヤー (未使用)
Direction	実数型	エージェントの向いている方向 (未使用)
state	整数型	node エージェントの状態を表す。 0 : 未確定ノード 水色の四角形画像で表示する。 1 : ワーキングノード ピンク色の三角形画像で表示する。 2 : 確定ノード 黄色の円形画像で表示する。
IDNo	整数型	プログラム上ノードを識別するための数値
link	エージェント集合型	node エージェントが連結する連結先 node エージェントを格納する変数。 自動的に青色の線が描かれる。
Tree	エージェント集合型	自分が起点で赤矢印を引く相手の node エージェントを格納する変数
LinkAgents	エージェント集合型	自分の連結情報を持つ link エージェントを格納する変数
TreeFrom	エージェント集合型	自分に向けて赤矢印を引こうとする相手の node エージェントを格納する変数。
color	整数型	ノードの表示色
icon	文字列型	表示画像名称 (この変数に画像名称が格納されると artisoc が自動的に node エージェント位置に画像表示を行う)

(2) link エージェント

node エージェントの連結情報を保持するために使われるエージェント。エージェントとしての表示・動作は無い。

変数値はエクセルシート上で作成し、CSV テキストファイル化したものを初期値として読み込ませている。

変数名	型	用途
ID	整数型	artisoc エンジンが自動的に各エージェント識別用に設定する値(未使用)
X	実数型	エージェントの X 座標値 (未使用)
Y	実数型	エージェントの Y 座標値 (未使用)
Layer	整数型	ノードを表示するレイヤー (未使用)
Direction	実数型	エージェントの向いている方向 (未使用)
FromNode	整数型	連結元 node エージェントの IDNo を保持する
ToNode	整数型	連結先 node エージェントの IDNo を保持する
cost	整数型	連結先 node エージェントへパケットを転送するコスト値

(3) name エージェント

node エージェントの近くでノード名称を表示するために使われるエージェント。

エクセルシート上で作成し、CSV テキスト ファイル化したものを初期値として読み込ませる。

変数名	型	用途
ID	整数型	artisoc エンジンが自動的に各エージェント識別用に設定する値(未使用)
X	実数型	エージェントの X 座標値
Y	実数型	エージェントの Y 座標値
Layer	整数型	ノードを表示するレイヤー (未使用)
Direction	実数型	エージェントの向いている方向 (未使用)
name	文字列型	ノード名称文字列 (この変数に画像名称が格納されると artisoc が自動的に name エージェント位置に文字列表示を行うように設定する)

3. シミュレーションのスクリプト

3. 1 ダイクストラ (Dijkstra) アルゴリズムの実装

単一始点最短パス問題を解くダイクストラの

アルゴリズムは、参考にした書籍では、(表 2.) のような疑似プログラム風に表現されている。これは、始点 s からグラフ上の全点への最短パスの距離を求めるアルゴリズムである。

```

 $(G, \ell)$  に対する Dijkstra のアルゴリズム :
コメント :  $S$  は確定済みの点の集合となる
        また、各  $u \in S$  に対して、 $s$  からの距離は  $d(u)$  として記憶される
 $S = \{s\}$ ,  $d(s) = 0$  と初期設定する
While  $S \neq V$ 
     $S$  からの辺を少なくとも 1 本もつ点  $v \notin S$  のうちで
         $d'(v) = \min_{e=(u,v): u \in S} \{d(u) + \ell_e\}$  が最小となる  $v$  を選ぶ
     $v$  を  $S$  に追加し  $d(v) = d'(v)$  とする
Endwhile

```

(表 2.) Dijkstra のアルゴリズム (『アルゴリズムデザイン』より⁷⁾)

(表 2.) で記述されたダイクストラのアルゴリズムを、今回作成したシミュレーションでの処理の実装と対比させて説明する。

アルゴリズムは、始点 s からの最短のパスの長さ (s からの距離) $d(u)$ が確定した点 u の集合 S を管理している。集合 S に属する点を確定ノードと呼ぶことにする。最初、 $S = \{s\}$ かつ $d(s) = 0$ と初期設定する。

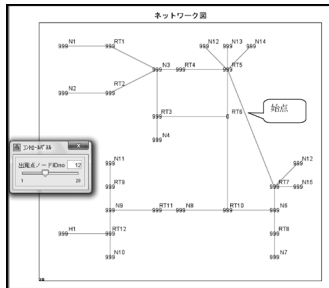
今回作成したシミュレーションでは、点をノード (node) と呼ぶ。シミュレーション起動直後に、全ノードにつき始点 s からの距離を

意味する整数型変数 $cost$ を 999、ノードの状態を表す整数型変数 $state$ を 0 で初期化する。 $cost$ 値 999 は、無限大・あり得ないほど大きなコスト値または、コスト値が未確定なノードのコスト値という意味で使っている。 $state$ 値 0 は、未確定ノードであることを意味する。また始点 s の $cost$ 値は 0 である。

(表 2.) における距離 $d(u)$ は、シミュレーションにおいては、ノード u の $cost$ 値と同じ意味である。

(図. 9) は、ノード RT6 を始点にしたシミュ

レーション開始直後の様子である。始点ノード RT6の距離（コスト値）は0、その他全ノードの距離（コスト値）は無限大値を意味する999に初期化されていることが分かる。（表2.）に



（図9.）シミュレーション開始直後の様子

おける、 V とは全ノード集合のことで、 $d(u)$ が確定した点 u の集合 S が V と一致するまで While ループ内の処理が繰り返される。

While ループ内では、距離（始点からのコスト） $d(u)$ が確定済みのノード集合 S に属する点 u からつながる辺をもつ点（集合 S に属さない未確定の点）のうちで、始点 s からの距離（コスト値）が最小となる点 v を選び、その点を新たに集合 S に追加し $d(v) = \text{始点 } s \text{ からの距離（コスト）}$ とする。

artisoc によって視覚的にシミュレーションの進行を追えるので、シミュレーション実装では、それに適応してアルゴリズムを少しアレンジしている。以下 (a) ～ (d) の処理によって実現している。

(a) (図10.) は、(図9.) から1ステップシミュレーションを進めた状態の様子である。

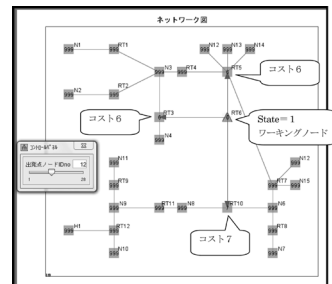
まず始点 s からの距離が最小（最小 cost 変数値を持つノード）となる未確定ノードを探す。最初は始点 s が $\text{cost} = 0$ の最小値を持つので、必然的に始点 s ノードが選択される。

(b) 選択されたノードは $\text{state} = 1$ にする。この状態をワーキングノード（作業中ノードと

いう意味）と呼ぶことにする。ワーキングノードはピンクの三角形で表示する。そして早々とこの段階で、ワーキングノードと連結している未確定ノードへの辺を赤い矢印で結ぶ。また、ワーキングノードと連結するノードの距離（コスト値）も計算しノードの cost 変数に書き込む。

(図10.) では、ワーキングノード RT6がピンクの三角形で表示され、それと連結している3つのノード RT5, RT3, RT10が赤い矢印で連結され、コストもすでに計算されて表示されている様子が窺える。

(c) 処理 (b). 終了後、ワーキングノード



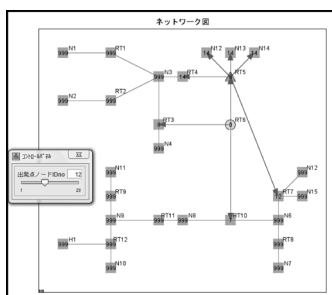
（図10.）シミュレーション起動後1ステップ進出した時点の様子

の $\text{state} = 2$ （確定ノード）にする。確定ノードは黄色の円形で表示することにした。

(図11.) は、(図10.) から1ステップシミュレーションを進めた状態の様子である。処理済みのノード RT6が黄色の円形で表示されていることが分かる。

(d) 以上 (a) ～ (c) の処理を未確定ノードがなくなるまで繰り返す。

(図11.) では、RT6ノードがワーキングノードになっている段階の後、未確定ノードでコスト値が最小である RT5がワーキングノードとして選択され、処理 (b) が実行されている様子が窺える。

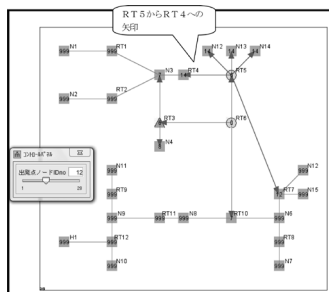


(図11.) シミュレーション起動後2ステップ進行した時点の様子

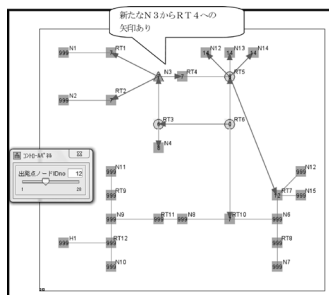
なお、始点 s から始まる赤い矢印の流れは、始点 s から全ノードへの最短経路ツリーを形成するものであるが、(表2.)のダイクストラのアルゴリズムに忠実に従うならば集合 S に属する確定ノード間でのみ引かれるべきものであろう。今回のシミュレーションでは、視覚的にわかりやすい表現にしたい都合から、確定ノードおよび、ワーキングノードからつながる全ての未確定ノードへの赤い矢印が早い段階で引かれている。(さらに未確定ノードであるにも関わらずコスト値も計算済みになっている)

このことは、赤い矢印が始点 s から全ノードへの最短経路ツリーの流れを意味するとともに、シミュレーションの途中では作業の進行状況の途中経過をわかりやすく表示している意味も持つ。従って、一旦引かれた赤い矢印が後に削除されるケースがある。また、未確定ノードのコスト値が変更されることもある。

例として、(図12.)では、すでにノード RT5 から RT4 への赤い矢印が引かれているが、シミュレーションをもう1ステップ進行させた(図13.)では、この矢印は取り消され、代わりに新たなノード N3 から RT4 の矢印が引かれている様子が窺える。



(図12.)



(図13.)

3.2 シミュレーションスクリプトの構成

artisoc のスクリプトは、独立したセクションに分かれている。

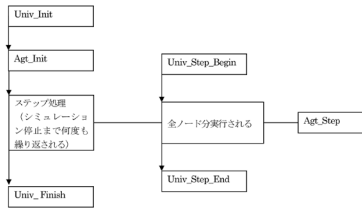
シミュレーションを実行すると、まず Univ_Init セクションに書かれたスクリプトが実行され、次にエージェントの初期化処理 Agt_Init が実行され、その後ステップ処理が繰り返される。

各ステップ処理においては、まず Univ_Step_Begin セクションのスクリプトが実行された後、各ノードの振舞いが記述された Agt_Step セクションのスクリプトが全ノード分実行される。そしてステップ処理の最後に Univ_Step_End セクションのスクリプトが実行される。

シミュレーションの最後には Univ_Finish セクションのスクリプトが実行される。

(図14.) がスクリプトの構成概略図である。上から下に向かう順序で実行されるが、四角の右にある処理部分は、繰り返し実行される部分

という意味で図示してある。



(図14.) スクリプトの構成外略図

今回のシミュレーションでスクリプトを書き込んだセクションは、Univ_Init、Univ_Stop_End、node エージェントの Agt_Stop セクションの3箇所である。link エージェントは初期値データを保持する用途で使用しただけなのでスクリプト記述はない。name エージェントも座標位置に文字列型 name 変数に保持されている文字列を表示するために用意したが、単にエージェントの定義を行うだけで artisoc エンジンが自動的に表示を行うのでスクリプト記述の必要はない。また、これらセクションに書き込んだスクリプトとは別に3つのサブルーチンを独立したテキストファイルとして作成して使用し

た。

3.3 Univ_Init セクション

全ての link エージェントを読み込み、node エージェントの連結関係データ (FromNode、ToNode ペア) を取り出し、連結元 node エージェントのエージェント集合型変数 link に連結先 node エージェントを格納する。この処理は昨年度作成した電子回路シミュレーションと同じである。

さらに node エージェントのエージェント集合型変数 LinkAgents に自分の連結情報を持つ link エージェントを格納する。これは後に、リンクのコスト情報を取り出して使うためである。

そして、全てのノードについて state= 0 (ノードの状態を未確定にする)、cost= 999 (ノードのコストに最大値) の初期化を行う。

最後に、コントロールパネルのスライダーと連動しているグローバル変数 startNode と同じ IDno を持つノードをワーキングノードにする処理を実行して終わる。これは、シミュレーションの始点を決める意味がある。

```

1 include "SearchAgt.inc"
2 include "SetTrigger.inc"
3
4 Univ_Init{
5   Dim link As Agtset
6   Dim node As Agtset
7   Dim one As Agt
8   Dim FromAgt As Agt
9   Dim ToAgt As Agt
10
11   MakeAgtset (link, Universe.Nodefield.link) // link エージェントのみ集める
12   For each one in link
13     if one.FromNode <> 0 then
14       SearchAgt (one.FromNode) // リンク元ノードを検索する
15       if CountAgtset (Universe.nodeAgtset) > 0 then
16         FromAgt= GetAgt (Universe.nodeAgtset, 0)
17         AddAgt (FromAgt.LinkAgents, one)
18         // リンク元ノードの LinkAgents 変数に link エージェントを追加する
19       SearchAgt (one.ToNode) // リンク先ノードを検索する
20       if CountAgtset (Universe.nodeAgtset) > 0 then
21         ToAgt= GetAgt (Universe.nodeAgtset, 0)
22         AddAgt (FromAgt.link, ToAgt)
23         // リンク元ノードの link 変数にリンク先ノードを追加する
  
```

```

23     end if
24   end if
25 end if
26 next one
27
28 MakeAgtset (node, Universe.Nodefield.node)  // node エージェントのみ集める
29 For each one in node
30   one.state= 0  // ノードの状態を未確定に初期化する
31   one.cost= 999  // ノードのコストに最大値を設定する
32 next one
33
34 setTrigger (Universe.startNode)  // 出発点ノードをワーキングノードにする
35
36 }

```

3. 4 Univ_Step_End セクション

state = 1 (ワーキングノード) にする。

全ての node エージェントから、state が 0 (未確定ノード) で cost が最小値のノードを探索し、該当するノードが存在しない場合は、シミュレーションを終了する。該当するノードが存在する場合はそのノードを

```

1 Univ_Step_End|
2 Dim minValue As Integer
3 Dim node As Agtset
4 Dim one As Agt
5 Dim minNode As Agt
6
7 minValue= 999
8 MakeAgtset (node, Universe.Nodefield.node)  // node エージェントのみ集める
8 For each one in node
10   if one.state == 0 then  // 未確定ノードあり
11     if one.cost < minValue then
12       minValue= one.cost
13       minNode= one
14     end if
15   end if
16 next one
17
18 if minValue == 999 then
19   ExitSimulationMsgLn ("Simulation Completed after " & GetCountStep () & "Steps")
20 else
21   minNode.state= 1  // cost が最小のノードをワーキングノードとする
22 end if
23
24 }

```

3. 5 node エージェントの Agt_Step セクション

node エージェントの状態を意味する state 変数の値によって表示を切り替えている。

(図15.) に示した state 変数値と同名の小さな画像 3 種類を表示に使用している。



(図15.)

またワーキングノードの場合には、ノードに連結する未確定ノードのコスト計算処理、ワーキングノードと未確定ノード間に赤矢印を引く

処理なども行っている。処理が終わった後は、に変更する。
ワーキングノードを確定ノード (state= 2)

```

1 include "SearchAgt.inc"
2 include "SetTree.inc"
3
4
5 Agt_Init|
6
7 |
8
9 Agt_Step|
10 Dim one As Agt
11 Dim linkCost1 As Integer
12 Dim link As Agt
13 Dim linkNode As Agt
14 Dim linkCost As integer
15
16 if My.IDNo <> 0 then      // 普通のネットワークノードの場合
17     if My.state == 0 then
18         My.icon= Cstr (0) & ".jpg"           // 水色四角形表示にする
19
20
21     elseif My.state == 2 then
22         My.icon= Cstr (2) & ".jpg"           // 黄色円形表示にする
23
24
25     elseif My.state == 1 then  // ワーキングノードの場合は忙しい！
26         My.icon= Cstr (1) & ".jpg"           // ピンク色三角形表示にする
27
28         For each one in My.link
29             For each link in My.LinkAgents
30                 if (link.FromNode == My.IDNo) and (link.ToNode == one.IDNo) then
31                     linkCost= My.cost + link.cost      // リンク相手の新コスト
32                     if one.cost > linkCost then  // 相手ノードのコストが大きい場合
33                         one.cost= linkCost      // 新しいコストに書き換える
34                         one.state= 0             // 相手ノードの state を未確定にする
35                         setTree (My, one)        // Tree 関係を構築する
36                     end if
37                 end if
38             Next link
39         Next one
40
41         My.state= 2      // 自分の state を確定にする（確定ノードとする）
42
43
44     end if
45 end if
46 |
47
48

```

3. 6 サブルーチン

れて使われる。

(1) サブルーチン SearchAgt

IDNo をキーに全 node エージェントから目

同名の SearchAgt.txt テキストファイルとし 目的の node エージェントを探す。
て保存して使用。

インクルード宣言 (include "SearchAgt.
inc") により artisoc により自動的に読み込ま

```

1 // IDNoを持つエージェントを全エージェントから検索する
2 Sub SearchAgt (IDNo As Integer) {
3
4   Dim allAgent As Agtset
5   Dim dest As Agtset
6
7   ClearAgtset (Universe.nodeAgtset)
8
9   MakeAgtset (allAgent, Universe.ElectricCircuit.node)
10
11  for each one in allAgent
12    if one.IDNo == IDNo then
13      AddAgt (Universe.nodeAgtset, one)
14      Break // もう調べる必要なし (エージェントが見つかった)
15    end if
16  next one
17 }
18

```

(2) サブルーチン SetTree

同名の SetTree .txt テキストファイルとして保存して使用。

インクルード宣言 (include "SetTree.inc") により artisoc により自動的に読み込まれて使われる。

サブルーチンの引数 Myself ノードから linkAgent ノードに赤矢印を引く処理を行う。すでに linkAgent への他の赤矢印が引かれていたなら、その赤矢印は消去し、新たな赤矢印を引き直す。

```

1 / 連結するノードとの Tree 関係を更新する
2 Sub SetTree (Myself as Agt, linkAgent As Agt) {
3   Dim one As Agt
4   Dim TreeFrom As Agtset
5
6   TreeFrom = linkAgent.TreeFrom
7   For each one in TreeFrom
8     RemoveAgt (one.Tree, linkAgent)
9     // これから Tree 関係を作るノードと Tree 関係になっているノードを外す
10  next one
11
12  ClearAgtset (linkAgent.TreeFrom) // これから Tree 関係を作るノードへの古い TreeFrom を消す
13
14  AddAgt (linkAgent.TreeFrom, Myself) // 連結するエージェントに自分からの Tree 関係を作る
15  AddAgt (Myself.Tree, linkAgent) // 自分の Tree に連結するエージェントを加える
16
17
18 }

```

(3) サブルーチン SetTrigger

同名の SetTrigger .txt テキストファイルとして保存して使用。

インクルード宣言 (include "SetTrigger.

inc") により artisoc により自動的に読み込まれて使われる。

サブルーチンの引数 IDno を持つノードを探して、そのノードをワーキングノードにする。

```

1 // トリガーエージェントの状態をワーキングノードにする
2 Sub SetTrigger (IDNo As Integer) {
3
4   Dim trigger As Agt
5
6   SearchAgt (IDNo)

```

```

7
8 if CountAgtset (Universe.nodeAgtset) > 0 then
9
10 trigger= GetAgt (Universe.nodeAgtset, 0)
11 trigger.state= 1 // 状態をワーキングノードにする
12 trigger.cost= 0 // コストを0にする
13
14 end if
15

```

4. 結び

今回作成したシミュレーションも昨年度の電子回路シミュレーションと同様に、ルールの総計が200行までという制限のある artisoc textbook で作成可能であった。エクセルシート上で、node エージェント・link エージェント・name エージェントデータを作成し、artisoc に読み込ませ、ノード数に合わせてコントロールパネルに変更を加えれば、今回とは異なったネットワーク上でシミュレーションを動作させることができる。

学生各自が授業で使用するパソコンで artisoc が使える環境になっていれば、シミュレーションを自分で動かして、演習することが可能である。コントロールパネルのスライダーを操作することにより、始点ノードを学生各自が選択しシミュレーションを動かし、最終的に選択したノード（ルータ）の経路制御表を作成するまでの演習ができる。

演習時には、まず経路のコストが書き込まれた図（図5.）を資料として学生に配布して説明する必要がある。

そして、以下の①～③の説明が必要であろう。

- ①ネットワーク上には28個のノードがあること。
- ②コントロールパネルのスライダーを操作して変更可能な出発点ノード IDno 値は、各ノードが持っている IDno と対応しており、（図5.）上でノードの近くにある丸数字が、そのノードの IDno を表していること。選択したいノードを見つけるには、丸数字を参照すればよい。
- ③28個のノード全てを始点として最短経路を探

索することができるが、最終的に経路制御表を作るのが目的であれば、最短経路探索なしでも簡単に作成できてしまうノードがある。それらのノードは演習には使わないこと。

（図5.）では、以下の（表3.）ノードの8個のうちから選択して演習することになろう。

IDno	ノード名称
3	N3
4	RT4
8	RT5
11	RT3
12	RT6
17	RT7
22	RT10
23	N6

（表3.）

（表3.）の8つのノードのいずれかを選択してシミュレーションを動かし、選択ノードを起点とした最短経路ツリーを作成し、（図7.）のような手書きのツリーを書く。最終的に選択ノードの経路制御表を作成する。

シミュレーションと実際の OSPF ネットワークルーティングとは合っていない点もあるが、ネットワークルーティングの基本的な動作を確認してもらう演習にはなるであろう。

今回のシミュレーションは OSPF ネットワークシミュレーションと言うよりも、OSPF ネットワークルーティングでルータがトポロジデータベースから最短経路ツリーを計算するアルゴリズム部分のシミュレーションと言う

べきであろう。

グラフに関するアルゴリズムを手作業で実例に基づいて実行し体験するのはかなり困難である。しかし、このような視覚的に動作が追えるようなシミュレーションを使えるとアルゴリズムの体験が容易であり、artisoc で簡単に作成できることが実感できた。artisoc はアルゴリズムの説明用シミュレーションを作成するのにも便利なツールのようなものである。また、アルゴリズム説明用と言うほどではなくても、情報処理講義用に、説明を補助するちょっとした、何か画像が切り替わって動くようなものを作るのも簡単である。今後は授業で使える補助的ツールも artisoc で作成していこうと考えている。

[参考文献・出典]

- 1) 末木俊之 (2008) artisoc で作成する初等情報教育用電子回路シミュレーション、駒沢女子大学研究紀要、p.87-102
- 2) 山影進 (2007) 人工社会構築指南 artisoc によるマルチエージェント・シミュレーション入門、有限会社書籍工房早山、東京
- 3) 小口正人 著 (2007) Computer Science Library 8 コンピュータネットワーク入門 - TCP/IP プロトコル群とセキュリティ、p.109、サイエンス社、東京
- 4) 小口正人 著 (2007) Computer Science Library 8 コンピュータネットワーク入門 - TCP/IP プロトコル群とセキュリティ、p.132、サイエンス社、東京
- 5) 小口正人 著 (2007) Computer Science Library 8 コンピュータネットワーク入門 - TCP/IP プロトコル群とセキュリティ、p.136、サイエンス社、東京
- 6) 友近剛史・池尻雄一・小早川知昭著 (2006) ネットワーキング入門シリーズ 2 インターネットルーティング入門、第2版、p.94、翔泳社、東京
- 7) J.Kleinberg・É.Tardos 著、浅野孝夫・浅野泰仁・小野孝男・平田富夫訳 (2008) アルゴリズムデザイン、p.125、共立出版、東京
- 8) 網野衛二 (2008) 3 分間ルーティング基礎講座、技術評論社、東京