

マイクロフィルム用資料

ISSN 0386—5878

土木研究所資料第3587号

土木研究所資料

震災対策体制のパフォーマンス評価手法に 関する基礎的研究

平成10年6月

建設省土木研究所
耐震技術研究センター防災技術課

Copyright © (1998) by P.W.R.I.

All rights reserved. No part of this book may be reproduced by any means, nor transmitted, nor translated into a machine language without the written permission of the Director General of P.W.R.I.

この報告書は、土木研究所長の承認を得て刊行したものである。したがって、本報告書の全部または一部の転載、複製は、土木研究所長の文書による承認を得ずしてこれを行ってはならない。

震災対策体制のパフォーマンス評価手法に 関する基礎的研究

防災技術課 課長 杉田秀樹
主任研究員 野崎智文

要 旨

地震発生直後、社会基盤施設を管理する組織は、広範にわたる所管施設の被害状況を迅速かつ的確に収集することが求められる。このような体制は、通常多くの体制要員や情報伝送・処理等を行う機器の集合体であり、それらの要素の連携によって情報が収集され、集約されてゆく。ところで、体制全体としての情報管理の迅速性・的確さ（すなわちパフォーマンス）は、個々の人・機器のみの効率を高めるだけでは必ずしも実現されず、それらの要素が有機的に連携した「システム」として評価されなければならない。本研究では、震災対策体制において被害情報が収集される迅速性を体制のパフォーマンスとし、体制をネットワーク表現することによって、そのパフォーマンスを定量的に評価する手法に関する基本的な定義、設定、適用可能性について検討した。

キーワード：震災対策体制、震前評価、情報管理、ネットワーク解析、オブジェクト指向

目 次

1. 序 論	1
2. 手法の概要	5
2.1 震災対策体制のフェーズ分析と要素処理の分析	5
2.2 ネットワークによる体制の表現	6
2.3 手法の特徴	7
3. 手法の詳細	8
3.1 情報量の定義	8
3.2 体制構成要素のゲート	9
3.3 ネットワーク全体のパフォーマンス評価の方法	11
3.4 ネットワークの操作	12
4. ネットワーク表現のアルゴリズムの実現方法	16
4.1 オブジェクト指向によるアプローチ	16
4.2 ネットワーク操作の方法	19
4.3 具体的なコード例	20
4.3.1 クラス宣言のコード	20
4.3.2 クラスの操作法	22
5. モデル体制によるケーススタディー	24
5.1 モデル体制の設定およびネットワーク表現	24
5.2 試算ケースの設定	27
5.2.1 ケース・パターンの設定	27
5.2.2 パラメーター設定	29
5.3 結果	30
6. 結 論	36
参考文献	37
附 図	39
ソース・コード	63

1. 序 論

世界有数の地震多発国である日本では、これまで多くの大規模地震を経験してきた。そしてそのたびに新しい知見を得て、地震に強い国づくりを進めてきた。従来、地域の安全性を確保するための方策として、生活基盤を形成する建築物・公共土木施設の耐震性を高めることに意がはられてきた。このようなハード面からの対策は、いうまでもなく地震に対する地域の安全性を確保するのに必須の条件である。しかしながら、希な確率で発生する巨大な地震に対して「絶対に安全な」施設を設計し施工することは、コストの面から見ても現実的ではないことがありうる。したがって最近では、地震によって施設に被害が発生することを想定し、地震直後の緊急対応活動を効率的に行うことによって施設被害の影響を最小限に押さえるための対策も重要となっている。このような対策においては、地震発生直後に施設の被害状況をあらかじめ把握できていれば効率的な活動を行うことができるが、そのためには施設の被害を直接検知したり、オンライン地震計から送られるデータによって施設被害を迅速に推定することが考えられ、その結果によって、効率的な点検等の活動に対する判断材料を得ることができる。

一方、都市の発達とともに人の動きやこれを支えるシステムはより複雑になり、災害に対する脆弱性、波及の影響は増大している。したがって、実際に地震が発生し被害が生じた場合にその影響を軽減するために、被害情報の収集・伝達技術、緊急活動のための有効な震災対策体制のあり方を充分検討しておく必要がある。一般にこのような体制は、情報収集、点検などの機能ごとのグループによって編成されることが組織の防災計画等¹⁾に示されているほか²⁾、連絡系統、参集・点検方法、報告様式などがマニュアル化されていることが多い³⁾。またこのような体制活動を支援するための各種の情報機器・システムによって、体制活動をさらに効率化するための技術開発が進められている⁴⁾⁵⁾。前述の検知、モニタリング、推定システムもこの範疇に属する。ところが、このような体制においても、以下のような問題点が内在している。

- ① 十分高速な機器システムを有していながら、参集や現場への移動が不可能であったり長時間を要したりするために、緊急対策活動が遅延するのではないか。
- ② あるセクションが迅速に情報を収集したにもかかわらず、それを集計するセクションあるいは機器で相当の時間を消費するために、緊急対策活動が遅延するのではないか。
- ③ 地上点検による情報収集は立ち上がりが遅く、全容把握に時間がかかるため、ヘリコプターによる上空からの被害発見や、センサー等による被害検知が有効なのではないか。
- ④ その際に迅速性を確保し、また過大でない投資によって妥当な精度の情報を得るためには、必ずしも正確な情報ではなく、被害の存在の可能性あるいはある程度の規模を知るだけでも十分有効なのではないか。

従来の震災対策計画においては、体制の要員、活動班構成や報告様式といった規定については細

かく定められていたが、その体制が緊急時にどれだけの能力を発揮するのか、あるいはどのような問題点があるのか、問題点があるとすればそれはどこにあるのか、といったアセスメントを考慮していない。地震発生時に可能な限り迅速かつ的確に活動することが要請される震災対策体制については、単に体制を一意に規定し、それに対して訓練を行うのみならず、体制活動の迅速性、正確さといったパフォーマンスを評価し、複数の体制代替案について検討しておく必要がある。

本研究では、震災対策体制システムを要員、機器などが有機的につながったネットワークシステムとして捉え、この体制モデルについて情報管理のパフォーマンス（すなわち活動の速さ、情報管理の正確さ）を評価する手法について検討する。本研究における手法によって、平常時において、緊急事態を想定した体制管理を定量的な評価に基づいて行うことが可能となり、地震が発生した際の活動をより効率的なものとする事ができる（図 1.1）。

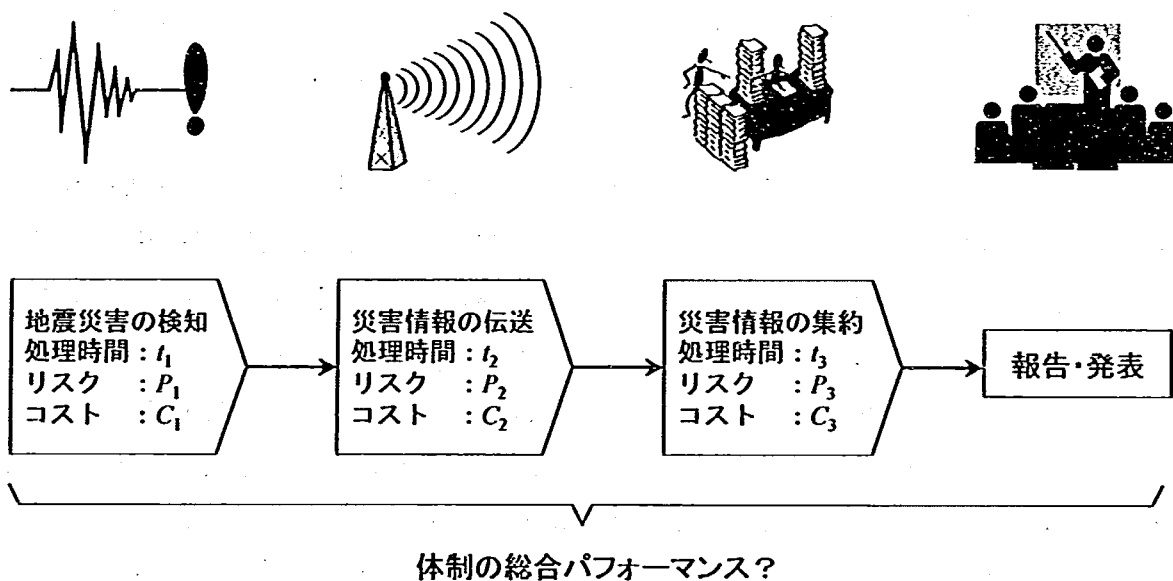


図 1.1 震災対策体制のシステムの表現・評価

地震発生時に情報を迅速に伝達するためのシステムのあり方については、たとえば片田らによって研究されている⁷⁾。これは災害時の被災地における住民間の情報伝播をシミュレートするものであり、情報伝達を人的ネットワークモデルで表わし、コミュニティとしての情報伝播の速さを評価している。この方法は、地域における災害警報・拡声器等、情報伝播のソースの配置計画に適用可能であり、ある意味で災害情報伝達システムのパフォーマンスを評価する技術と言える。ただしこの方法は、抽象的に意味付けされた「情報」が一般の住民の場において拡散してゆく様子をモデル化するものであり、震災対策体制のように組織化され、体制各要素が具体的な役割を持った複雑なシステムに対して適用する手法ではない。

一方、複雑なシステムをネットワークにより表現し、情報等の流れを表現する手法が木俣らによって紹介されている⁸⁾。この方法では、災害対策組織体制、交差点における交通流といった複雑な事象を抽象的なネットワーク（ペトリネット）によって表現し阻害要因、遅延時間を特定しようとしている。しかしながらこの方法は表現が高度に抽象的で、震災対策現場において導入する際に計画者と解析者のインターフェイスがとりにくい。

震災対策体制といった組織的なシステムが全国の様々なレベルで行われている場合、それらの体制を評価し、より高度化するためには、直感的に把握しやすく、かつシステマティックな動作を表現することができる手法が必要である。このような問題意識のもと、著者らは震災対策体制の迅速性を定量的に評価する方法について研究開発を始めている⁹⁾。その過程では、震災対策体制の情報収集を対象とし、センサーによる被害検知、ヘリコプターによる上空からの点検、職員による点検などの方法の迅速性を定量的に評価した。この方法では、各被害検知方法ごとに、移動・点検といった時間を要する事象を接続し、モンテカルロ法によって多数の地震を発生させ、被害検知に要する時間を算出している（図 1.2）。

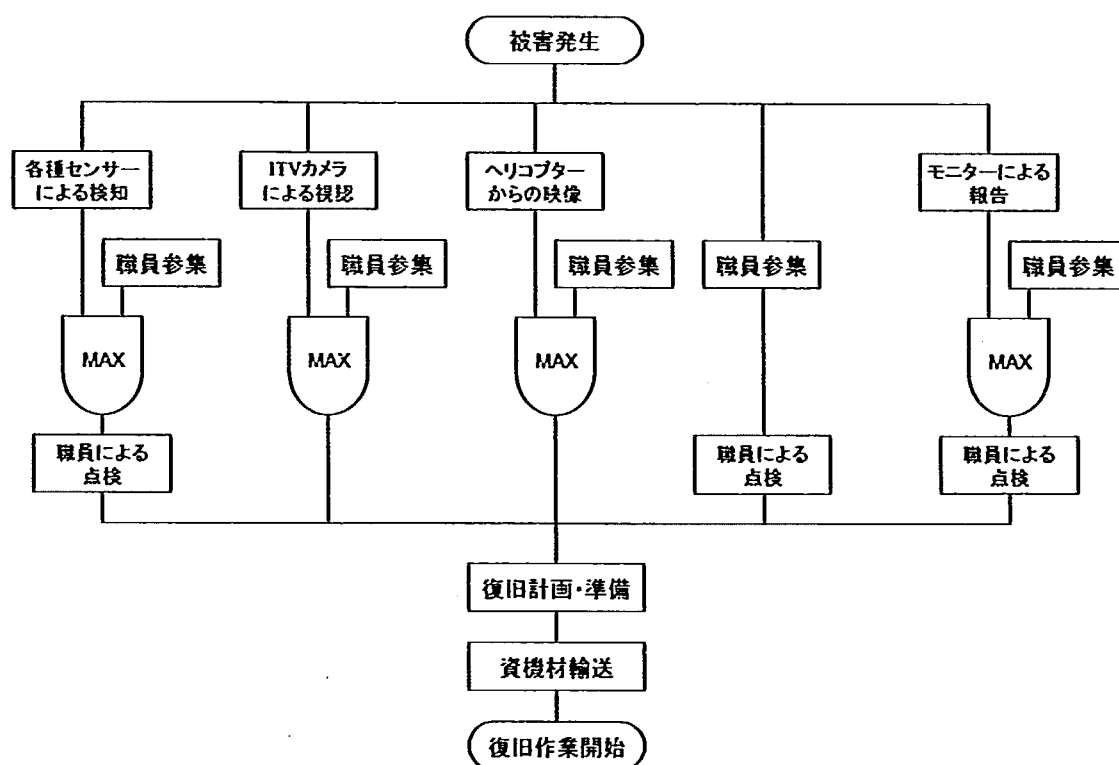


図 1.2 被害検知のパフォーマンス評価におけるネットワーク例

しかしながら、図 1.2 のような表現は震災対策における処理等を単純化した流れで表現した方法であって、具体的な人・機器の動作、反応、移動といった体制の構成要素の活動を正確に捉えたもので

はない。本研究ではこのようなアイデアをさらに進め、震災対策体制を人・機器といった構成要素が互に関連しながら具体的な動作をするネットワークとして表現し、情報収集の迅速性等のパフォーマンスを評価する手法を開発した。

まず第2章において、新しいアイデアである本手法の概念について述べる。続く第3章では、手法を組み立てるにあたって定義が必要な事項等について詳細に論ずる。今回検討を行った手法は、単に解析的な表現にとどまらず、その発想からシミュレーション・モデルによる実現にいたるまで「オブジェクト指向」によるアプローチを基本としている。第4章では手法によって具体的な体制ネットワークを構築する際の技法について説明する。考案した手法をモデルとして想定された工事事務所に適用し、その動作を検討した結果は第5章で示され、第6章で結論をまとめる。

2. 手法の概要

2.1 震災対策体制のフェーズ分析と要素処理の分析

複雑な震災対策システムをネットワーク表現するために、地震発生から被害点検結果が集約されるまでの組織の活動をいくつかのフェーズに分解する。たとえば河川・道路といった広域にわたる施設を管理している事務所における地震発生後のフェーズは概ね図 2.1 のように表せる。図中、「複数ルートでの点検」は、河川・道路等の複数の点検ルートに対して人員が別れて点検活動を行うことを意味する。すなわちここでは、地震発生の後それが検知され、必要に応じて職員が参集し、点検・報告された情報が集約される、という流れを想定している。なお、この点検の途上で点検箇所ごとに報告が行われる場合には、点検フェーズと情報集約フェーズが時間的に重なり合う。すなわちここでいう「フェーズ」とは、時間の流れに沿った区分ではなく、活動の質の観点からある程度一括して扱うことができる区分といった意味である。

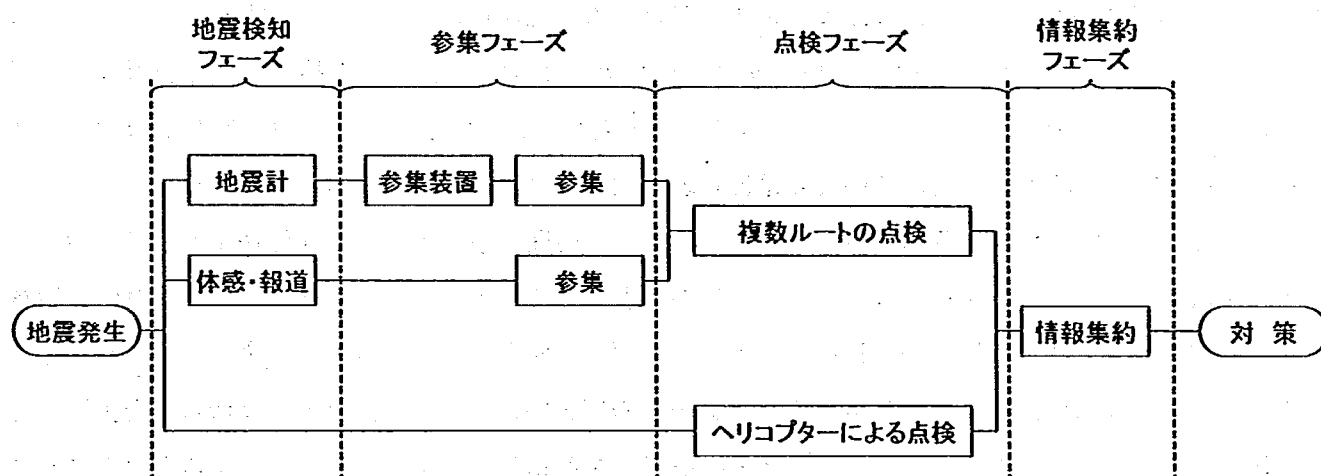


図 2.1 震災対策のフェーズ分析の例

各フェーズの項目は「地震計」、「複数ルートの点検」などに表示されているが、これらはさらに細かい経路、作業に分けられる。たとえば複数ルートの点検は、点検すべき路線に応じた経路に別れ、その中に点検すべき箇所数に応じて「移動」、「点検」、「報告」といった処理が含まれている。ここで「処理」とは、前段の処理の終了や別途の指示等によって始まり、一定の時間を経て後段の処理が可能となる体制中の要素で、たとえば人による被害結果の集計や意思決定、被害現場への移動、被害検知データの伝送、被害推定システムの実行など、体制中およそ時間を要するものは人、モノ、事象に関わらずこれに含めるものとする。たとえば先の移動、点検、報告という流れの一部を取り出してそ

の処理を模式的に表わすと、図 2.2 のようになる。

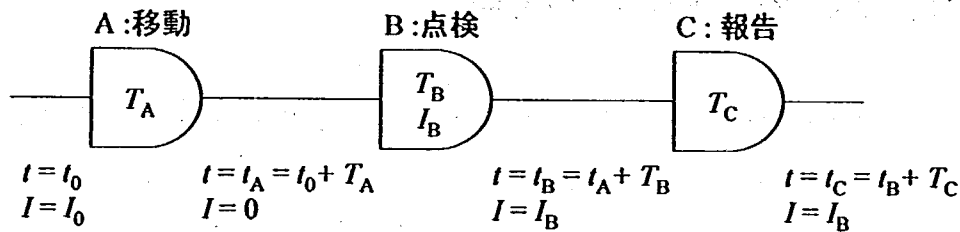


図 2.2 体制中の要素の模式的表現

図 2.2 において、移動開始前の時刻は t_0 で、この時点で前の個所の被災情報量 I_0 を有していたとする（前の個所の被災情報 I_0 は、すでに対策本部等に報告されているとする）。まず点検班は新しい点検箇所へ移動しなければならないが、これに時間 T_A を要するとする。この処理を表わしたのが「A:移動」の部分である。被災情報は点検後に随時報告し点検活動中に記録する必要がない場合には、移動後次の個所に到着した場合には被災情報 I がゼロあるいは任意の値となってもかまわない。またこの時、時刻は $t_0 + T_A$ となっている。点検活動には時間 T_B を必要とし、また点検によって個所 B の被災情報量 I_B が得られるとする。この処理は「B:点検」で表わされている。点検後の時刻は $t_B = t_A + T_B = t_0 + T_A + T_B$ 、情報量は I_B となる。その後の報告処理も同様に「C:報告」で表わされている。

図中 A、B、C はすべて入力を持ち、それに対して、時間、情報量の必要な処理を施して出力へ渡す。本研究ではこれらの処理を「ゲート」と呼ぶことにする。また、複数のゲートの前後関係に留意し、実際の作業、移動、情報の流れを表わすようにゲート間を接続したものを「ネットワーク」とする。図 2.2 は非常に簡単ではあるが、ネットワークの例といえる。なお図 2.2 では情報量を箇所ごとに報告し、クリアするとしたが、点検活動中に情報量を蓄積し、最後にこれを報告するというスタイルも、もちろん表現可能である。

2.2 ネットワークによる体制の表現

前節では、「移動」、「点検」、「報告」という三つのゲートを直列に接続したが、これよりも複雑な関係もありうる。たとえばあるセクションがあって、前段の二つの作業が終わってから後段の作業が始まり、所定の作業時間を経た後次のセクションの作業が可能になるような場合、図 2.3 のようなネットワークとなる。

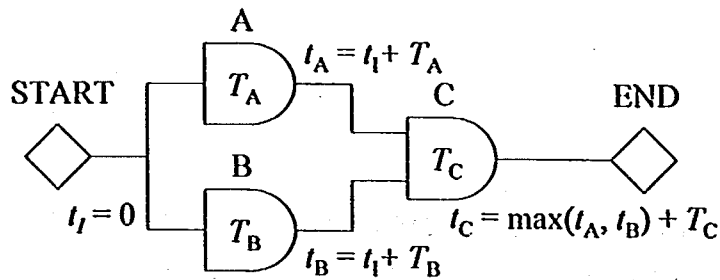


図 2.3 簡単なネットワークの例

この場合、ゲート A、B では作業時間 T_A 、 T_B を必要とし、ゲート C では A、B の遅い方を待って、その後作業時間 T_C を経た後、全体の処理が終了することを表している。ただしここでは情報の量については考えず、「情報あり／なし」のトリガーによって各ゲートが作動することとしている。

このようなネットワークを、震災対策体制の実際の処理に応じて表現し、情報集約のスピードと情報量の多さや正確さを集計することによって、体制のパフォーマンスを定量的に評価することが可能となる。

2.3 手法の特徴

本研究において検討する手法の概要は、次のようにまとめられる。

- ①震災対策体制における一定の役割あるいは機能を持った人・グループの移動や作業あるいは機器等の動作を、それぞれの所要時間を持った「ゲート」として表現する。
- ②ゲートは入力と出力を持ち、そこを情報が伝わると考える。
- ③地震発生をトリガーとして、各ゲートが順次作動し、箇所の点検等によって、体制全体として情報量が蓄積されてゆく。
- ④最終のゲート（たとえば情報集約班など）において得られる時間と情報の蓄積量の組み合わせによってパフォーマンスを評価する。

3. 手法の詳細

3.1 情報量の定義

震災対策体制中のゲートは、入力された情報を処理し、これを出力するという点で同一である。本節では、まずネットワーク表現された体制中を流れる情報の表現方法について定義する。まず情報 S は、それを観測する時刻 t とその時得られた情報量 I によって構成されと考える。

$$S = (t, I) \quad (3.1)$$

情報量 I については、①施設被害の点検結果が実際の被害を正確に表している程度、②施設被害の程度そのものあるいは③それらを複合化して数値化したものなどいろいろな定義が考えられるが、ここではこれを施設被害の点検結果の確からしさと定義する。たとえばある施設の被害をヘリコプターによって上空から点検する場合を考えると、施設の被害の属性を「大」、「小」、「なし」とした場合、遠方から確認するために実際の被害とは異なる判断をする可能性がある。このことを表したのが表 3.1 である。

表 3.1 ヘリコプターによる点検に関する情報量

		実際の被害		
		大規模	小規模	なし
観測情報量	被害大	1.0	0.0	0.0
	被害小	0.0	0.5	0.5
	被害なし	0.0	0.5	0.5
正答		1.0	0.5	0.5
誤答		0.0	0.5	0.5
過大 過小	過大	0.0	0.0	0.5
	過小	0.0	0.5	0.0

たとえば実際の被害が大規模な処置を要する大被害（たとえば落橋など）であった場合、たとえ上空のヘリコプターからであっても「被害大」と観測することはほぼ確実であるため、情報量は 1.0 とされている。一方、この被害を「被害小」、「被害なし」と見積もることはまずないことから、各々 0.0 を与えている。被害が小であった場合には、上空からこれを「被害大」と観測することはないので 0.0 としているが、これを「被害小」と見積もる場合と「被害なし」と見積もる場合が半分ずつの程度で起こりうると考え、各々に情報量 0.5 を与えている。被害なしの場合についても同様の考えである。これで実際の被害程度と観測された被害程度の組み合わせに対する情報量が定義されたが、最終的には、実際の各被害程度に対して情報量の一つ与える必要がある。この方法として、(a)被害を正しく見積もる可能性をもって情報量とする場合（表中の「正答」の欄）、(b)誤って見積もる場合（「誤答」の欄）、(c)被害を過大あるいは過小に見積もる場合（「過大」あるいは「過小」の欄）などいくつか

考えられる。ここでは「正答」の場合を情報量として考えることにする。

地上における点検の場合について同様の表を表 3.2 に示す。地上点検の場合、実際の被害を正確に点検すると考えられることから、すべての被害について情報量は 1.0 となっている。

表 3.2 地上点検に関する情報量

		実際の被害		
		大	小	なし
観測情報量	必要な処置	大規模	小規模	不要
	被害大	1.0	0.0	0.0
	被害小	0.0	1.0	0.0
	被害なし	0.0	0.0	1.0
正答		1.0	1.0	1.0
誤答		0.0	0.0	0.0
過大過小	過大	0.0	0.0	0.0
	過小	0.0	0.0	0.0

ここでは情報量の定義を観測された被害程度の確からしさという一点で評価した。これは施設被害の程度に関わらず点検結果の自信の程度(Confidence)を表わすものであるが、実際の体制においては被害の程度も情報量として重要な意味を持つ。

3.2 体制構成要素のゲート

(1) 最早動作ゲート

本節では、震災対策体制中の処理を表現するゲートを列挙する。まず複数の情報入力を持ち、ある処理時間で情報量を加工して出力するゲートを考える。最早動作ゲートは、複数の入力のうち最初の情報が到達した段階で処理を行い、結果を出力するゲートである (図 3.1)。この場合情報量の出力値は、時刻 t_i が最も小さい情報の情報量 I_i に対して施されるように表わされている。処理 f については、入力を変換するいろいろなパターンが考えられる。極端な場合、被災箇所の点検のように入力に関係なくあらかじめ設定された情報量が出力される場合には、処理 f は定数関数でもかまわない。

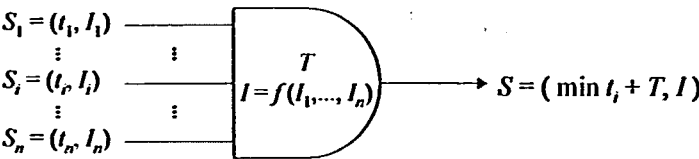


図 3.1 最早動作ゲート

(2) 最遅動作ゲート

最早動作ゲートとは逆に、複数の情報のすべてを待ってその後処理を行うゲートを最遅ゲートとする（図 3.2）。図中、処理 f としては、 n 個の情報量の最後の値、 n 個の情報量の合計値あるいは平均値など、演算上は様々なものが考えられる。

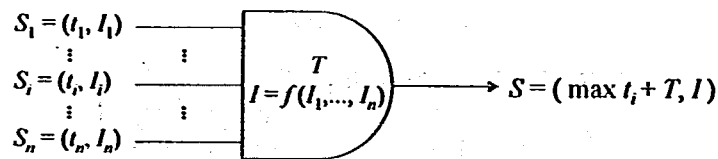


図 3.2 最遅動作ゲート

これらのゲートはその下位に属するゲートを包含している。たとえば入力の一つだけ持ち、入ってきた情報の時刻のみに反応して一定の処理時間後にトリガーのみをかけるゲートは、最早動作ゲートの入力の一つとし、任意の処理 f を定義すれば表現できる。一方、これらのゲートを拡張して新たなゲートを構築してもよい。たとえば最遅動作ゲートの内部に時刻の閾値を持たせその制限時間内に選ばれた入力のみに基づいて出力を計算するようなゲートも考えられる。

(3) 情報集約ゲート

情報集約ゲートは、各被災箇所からの情報が集約される集約班のイメージである（図 3.3）。このゲートは入力をそのままリストに蓄積していくもので、ネットワーク全体の分析が終わった段階で、リストには時刻、情報量のセットである情報が順不同に蓄積されている。後からこれを時刻順でソートするなどして、パフォーマンス評価に用いることができる。

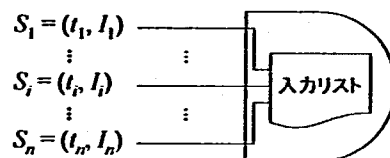


図 3.3 情報集約ゲート

(4) 地震発生ゲート

ネットワークの処理が開始される最初のゲートで、スタート・ノードと呼んでもよい（図 3.4）。このゲートは、後述するネットワーク操作の際に、ネットワーク動作の起点を示すものとして必要となる。同様にネットワークの終点を表わすエンド・ノードも必要となる。



図 3.4 地震発生ゲート

上記で紹介したゲートのほかにも、複数入力情報を待ち行列によって振り分けるゲート、外部からの指示等を表わすトリガーによって処理を行うゲートなど様々なゲートが考えられるが、今回のケーススタディーでは上記(1)～(4)のゲートを基本型としてモデル事務所を表現した。

3.3 ネットワーク全体のパフォーマンス評価の方法

多数のゲートが接続されたネットワーク・システムでは、被災箇所等の情報が処理され、流れて行き、最終的に情報集約ゲートのリストに情報が蓄積されてゆく。この情報を時刻の順に並べ、情報量を累積すると図 3.5 のようになる。これは、地震発生時点($t = 0$)以降震災対策体制において明らかになってゆく被害情報の総量を表わし、体制のパフォーマンスを定量的に表わしたものと見える。

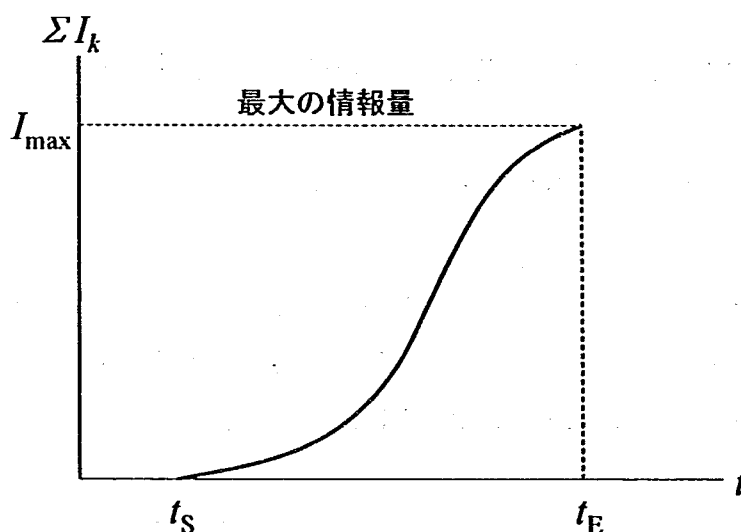


図 3.5 情報量の蓄積状況

図中 t_s は、地震発生後の混乱期を経て、各個所の被災状況が報告され始める時点である。その後、各個所からの被災情報が蓄積されてゆき、時刻 t_E の時点で、管内の全箇所が被災状況が正確に把握できたことになる ($I_{\max} = \text{箇所数} \times \text{情報量 } 1.0$)。地震直後すぐに点検活動を開始できる体制（すなわ

ち立ち上がりが速い体制)においては t_s が左に移動する。また、情報の収集効率がよい体制においては曲線の傾きが急になり、全体の被害状況を把握し終える時刻 t_e が右へシフトする。

震災対策体制において、効率的な参集システムを導入したり、点検ルートあたりの点検人員の構成を変更するなどの操作によってネットワーク形態が変化し、このパフォーマンスは変化するが、その結果を見ながら、平常時において有効な体制計画を立てることができる。

3.4 ネットワークの操作

本節では、多数のゲートを接続したネットワークを動作させる方法について検討する。再び図 2.3 に示した三つのゲートからなるセクションを考える (図 3.6)。

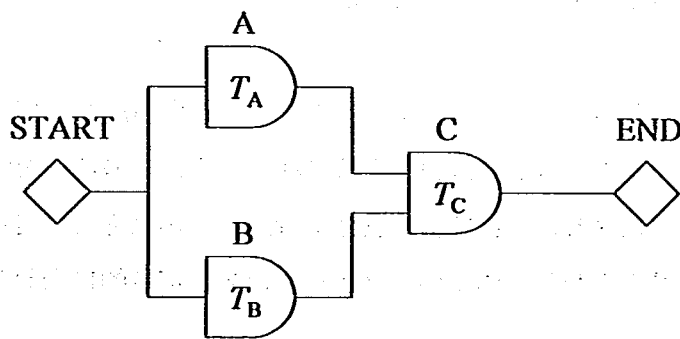


図 3.6 ネットワーク操作の説明のためのモデルネットワーク

ここで、ゲート A~C が次のような機能を持っているとする。

あるゲートに出力の要求があった場合、そのゲートが自動的に自分の入力をチェックし、必要な入力が全部そろっていれば自分の処理をそれに加えて出力を計算する。ただし、必要な入力がない場合、①さらにそのゲートが入力側のゲートに出力を要求する場合と②単に計算を保留する場合を考える。

このような性質を持ったゲートが接続されたネットワーク全体の処理を実行するアルゴリズムについて考える。

(1) 再帰法

まず考えられるのは、各々のゲートの出力側からネットワークの上流方向に向かってサーチしながら

ら計算してゆく方法が考えられる。ところがあるゲートの入力が必要なだけそろっていない場合、このゲートは自身の出力を計算することができない。そこで、入力が不完全な場合には自分の処理をいったん先送りしておいてから、その入力を得るために必要な前段のノードの処理を行うこととする。これは先のゲートの機能の後半の①に対応する。もし前段のノードでまだ出力が足りなければ、さらに同じ事を繰り返し、最終的にはスタート・ゲートの開始トリガーが得られるので、これまでたどった道筋を逆に帰ってゆき、すべての入力を得られるまでこれを繰り返す。このような処理は一般的には再帰的(recursive)な方法と呼ばれる。たとえば図 3.6 のネットワークについてこの動作をトレースしてみる。

- ① まずエンド・ノードがゲート C に出力を要求する (図 3.7(a))。
- ② 出力を要求されたゲート C は出力の計算をしようとするが、このゲートの入力は二つとも未定なので、そのうちの一つであるゲート A に出力を要求する (図 3.7(b))。
- ③ 出力を要求されたゲート A は出力の計算をしようとするが、これもまだ入力が定まっていないため、入力先のスタート・ゲートに出力の要求をする (図 3.7(c))。
- ④ スタートノードは要求に応じて、 $t=0$ を出力する (図 3.7(d))。
- ⑤ ノード A は入力 $t=0$ と自身の処理時間に基づき、出力 $t=T_A$ を出力する (図 3.7(e))。
- ⑥ ノード C は入力 $t=T_A$ を得るが、もう一方の出力が未定なため、ノード B に対して出力を要求する。このあと、上記と同じ流れでスタートノードまで溯り、ノード C のもう片方の入力 $t=T_B$ を得る。この段階でノード C の出力 $t=T_A + T_C$ が定まる (図 3.7(f))。そしてエンドノードはこの入力を受け取り、パフォーマンス評価のために保持する。

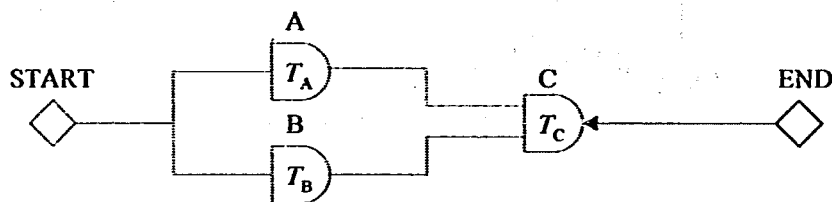


図 3.7(a) 最終出力ノードのゲート C への出力要求

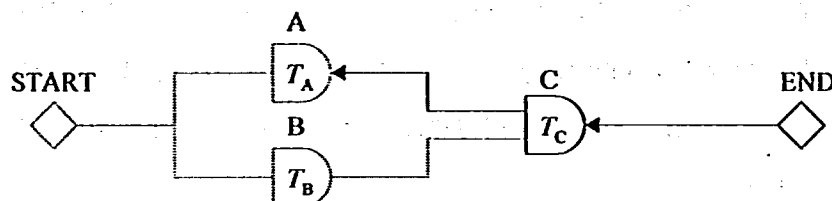


図 3.7(b) ゲート C によるゲート A への出力要求

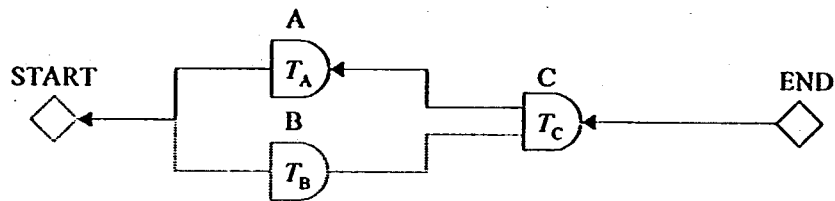


図 3.7(c) ゲート A による START ゲートへの出力要求

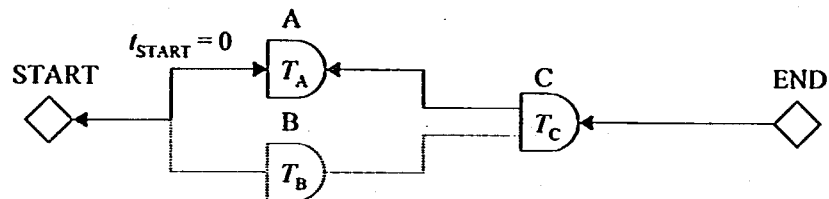


図 3.7(d) START ゲートの出力確定

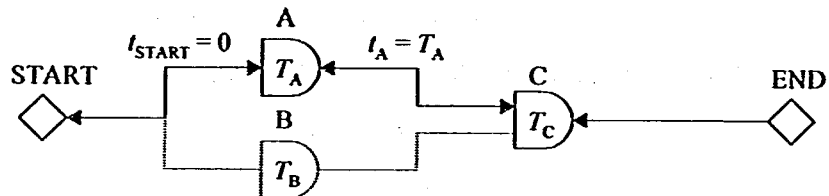


図 3.7(e) ゲート A の出力確定

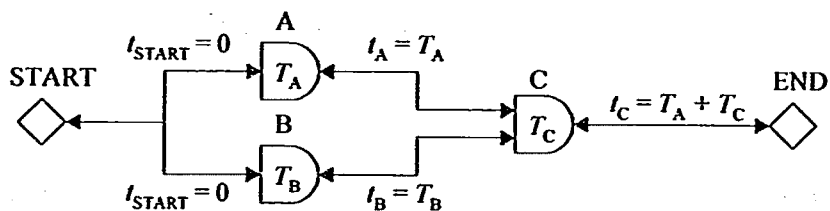


図 3.7(f) ゲート C の出力確定

(2) 総当たり法

もう一つの方法は、ネットワーク中の各ノードに二状態(OPEN/CLOSE)のフラグを一つずつ設け、すべてのノードを（その順序を考慮することなく）サーチしながら、ノードのすべての入力が決まって出力が計算できたならフラグを OPEN とし、すべてのゲートでフラグが OPEN となるまで計算を繰り返す方法である。この方法は、先のゲートの機能としては後半の②に対応する。上と同じ例でこれを示すと次のようになる。

① まず最初の状態ではすべてのノードでフラグが CLOSE となっている（図 3.8(a)）。

② 次に、一回目のサーチを行う。仮にノードサーチの順番が C、B、A の順番であったとすると、まずノード C ではまだ入力確定していないから、フラグは CLOSE のままであるが、ノード

B と A については、スタートノードで入力を与えられるためフラグは OPEN となり、出力 T_B 、 T_A が得られる (図 3.8(b))。

- ③ 二回目のサーチでは、ノード C の入力が得られているためその出力を計算する。ノード B と A についてはすでにフラグが OPEN なので無視する。これですべての出力が求まった (図 3.8(c))。

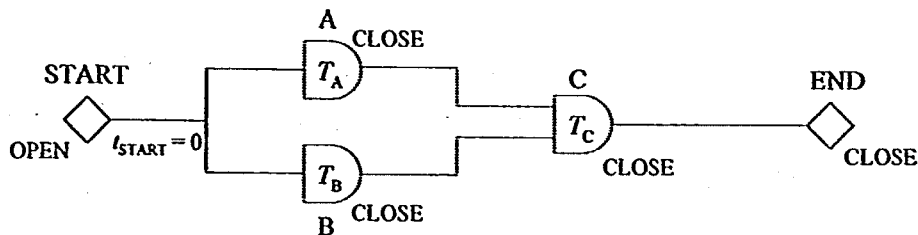


図 3.8(a) START ゲートの出力確定

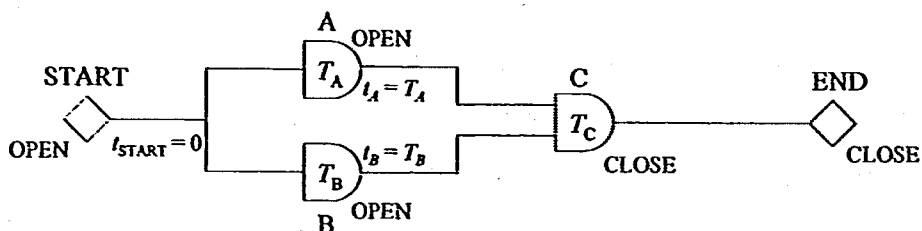


図 3.8(b) ゲート A、B の出力確定

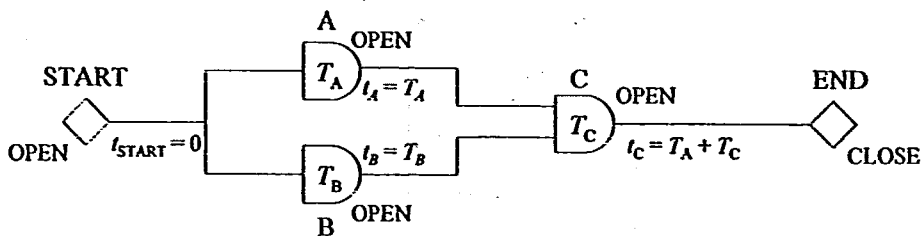


図 3.8(c) ゲート C の出力確定

なお、この例では意図的にゲート・サーチの順番を C、B、A としたのでサーチに二回を要したが、この順番が A、B、C の場合には、一回目のサーチですべてのゲートの状態が定まってしまう。このサーチの順番は、ゲートが生成・登録された順番によってのみ決まる。これらの方法のプログラム上の特徴については、次章で改めて説明する。

4. ネットワーク表現のアルゴリズムの実現方法

4.1 オブジェクト指向によるアプローチ

3.2 節で定義した多数のゲートを結合し、ネットワークとして取り扱うためには、ゲートを電子計算機の言語で表現し、シミュレートすることが有効である。このような開発をプログラミング言語として、FORTRAN、C といった従来の手続き型の言語を用いても実現可能であるが、ここでは特にオブジェクト指向の考えに基づいたプログラム設計を行う。従来のタイプの研究においては、理論とそのプログラム・コード化に密接なつながりはなく、主として理論面を重視した説明が多かったが、オブジェクト指向の考え方は、単なるプログラミング技法のみならず、その根本にある発想、理論の組み立てとも深く関係する。本章では、このオブジェクト指向に基づいて震災対策体制をネットワーク表現する方法について説明する。

(1) クラスの基本

オブジェクト指向のアプローチの特徴は、任意の属性・機能を有する「クラス」に基づいたオブジェクトにある。一般に、計算機内部のデータの表現として、整数型、実数型、文字型といった「データ型」が用いられるが、「クラス」とは、このような一般的な型を任意に組み合わせた構造体のデータ型に類似したものと考えてよい。たとえば、3.1 節で定義した「情報」の場合、時刻および情報量を表わす二つの実数型のデータから成り立っており、これに対して CInformation という「クラス」を定義することができる。

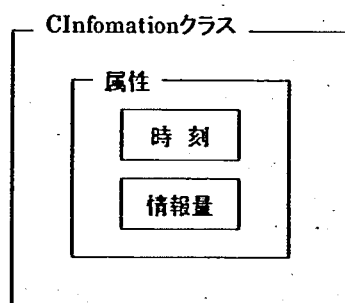


図 4.1 情報を表わすクラスの概念

このような考え方は、これまでも多くの言語において「構造型」データとして導入されてきたものであるが、クラスの特徴は、それに属性のみでなく機能も関連付けられることにある。たとえば CInformation クラスの入力と出力を一つずつ持ったゲートを CGate とクラスとして定義することを考える（図 4.2）。このゲートにはゲート特有の機能として「入力情報を用いて（経過時間や情報量など

の) 処理を行い、その結果を出力側で利用可能にする」という機能が必要である。

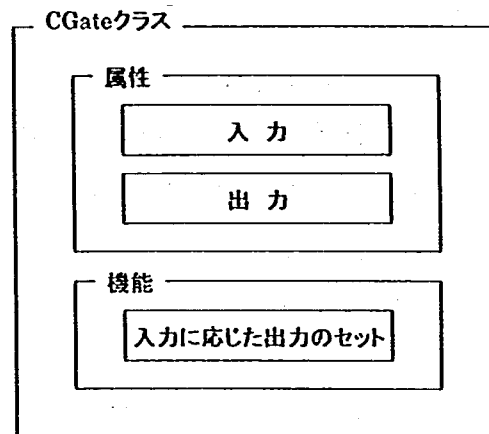


図 4.2 属性と機能を持ったクラス概念

ただしこの CGate クラスは、そのゲートでどのような処理を行うのか、処理に要する時間はいくらか、といった具体的な挙動がまだ定まっていない抽象的なもので、入力に応じて出力をセットする機能は具体的に定まっておらず、その存在だけが定義されている。

たとえば C++ 言語を用いた表現によって、クラスを利用する様子を表わすと次のようになる。まずあらかじめ CGate クラスが適切に定義されているとして、このクラスのゲートの一つ生成する。

```
CGate gate;
```

このとき、CGate クラスのオブジェクト gate が生成されたことになる。このゲートからの出力情報を利用する関数として GetOutput という機能 (関数) が定義されているとすれば、ゲートからの出力を取り出すための表現は以下になる。

```
gate.GetOutput();
```

同じような動作は、従来の手続き型言語でも

```
GetOutput(gate);
```

などとすることは可能であるが、後に述べる理由から、オブジェクト指向による考え方の方が有利である。このようなクラス固有の機能を実現する関数を「メンバー関数」と呼ぶこともある (同様にここでいう属性のことを「メンバー変数」と呼ぶこともある)。別のメンバー関数の例として、あるオブジェクトの入力を別のオブジェクトに接続する関数を定義することも考えられる。たとえばオブジェクト自身の入力と与えられたオブジェクトの出力を接続する関数が ConnectToOutputOf と定義されているとすると、その利用方法は次のような形が考えられる。

```
gate1.ConnectToOutputOf( gate2 );
```

ちなみにこのゲートが複数の入力を持つように定義されていれば、その宣言は

CGate gate(3);

とすることもできる。ここで'3'は三つの入力を持つゲートを宣言することを意味している。

(2) 派生(derivation)

CGate クラスは最も基本的なものであり、処理時間といったノード特有の性質を有していない抽象的なものである。ここでこのクラスと同じ性質を持ち、かつ入力～出力の間に一定の処理時間を持ったクラス CWaitGate を考える。入力の数を 3 個、処理時間を 5 でこのクラスのオブジェクトを生成するための宣言は、たとえば次のような形になる。

CWaitGate gate(3, 5)

このとき CWaitGate は、CGate の性質を持った上に CWaitGate 特有の性質と動作を有しており、プログラム時点で CGate を活用して少ない手順で新しい性質を持ったものを構築できる。このような関係を派生(derivation)といい、その関係を定義するのはプログラム上で行う。基本クラス CGate の方で、入出力といった属性や、任意の数の入力の確保、ノード間の接続、入力が決まっている場合の出力の計算といった手続きが定義されていれば、それから派生したクラスはその属性、手続きをすべて「継承(inheritance)」することができる。

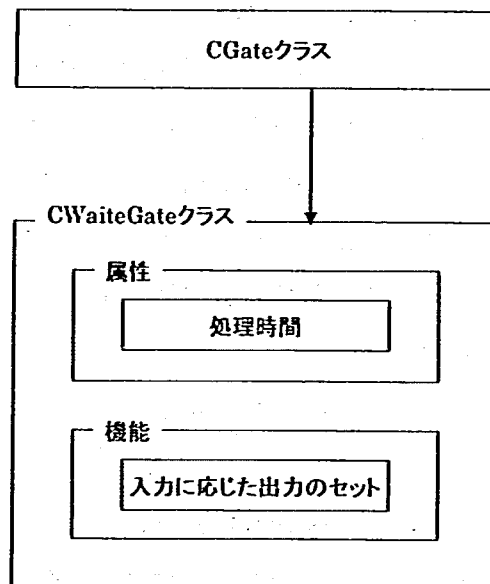


図 4.3 クラスの継承の概念

図 4.3 において、CWaitGate クラスにおいて新たに追加された属性である「処理時間」は CGate になかったもので、CWaitGate クラスに固有の属性である。一方、追加された機能「入力に応じた出力のセット」はベースの CGate にも存在するが、最初の時点ではこの機能は具体的に定められていなかった。新しいクラスの場合「入力情報の時刻に処理時間を加える」という、このクラス固有の具

体的な処理を行うことがわかっているので、これを具体的に定めることができる。このようなクラスの派生を応用すると、入力のうち時刻最大あるいは最小のものを利用するゲート、外からのトリガーに応じて反応するゲートなど、様々な機能を持ったクラスを派生させることができる。

(3) オブジェクト指向によるアプローチの利点

このように設計されたプログラムでは、任意に生成された個々のオブジェクトがあらかじめ設計されたクラスの手順どおりに作動する。すなわち各オブジェクトが自分の動作内容を「知って」いるため、それらを組み合わせた大規模なプログラムでも、FORTRAN などの手続き言語のようにメインプログラムあるいはサブプログラムでその構成要素のデータ構造に舞い戻って、手続きをいちいち考える必要がない。また新しい機能を持ったクラスが必要になったときに、所要の機能を有するクラスを構築あるいは派生させて、追加された手続きを規定するだけでよい。そのため、個々のクラスの動作に集中して設計することができる。またこのことから個々のクラスの設計とそのクラスを用いるプログラムの設計を切り離して行えるため、デバッグが容易であるといった利点がある。震災対策体制をネットワーク表現する場合、ネットワークを操作するメイン・プログラムでは個々のゲートの接続状況に応じた判断・演算を行う必要がなくなり、コード自体が非常に簡潔になる。

4.2 ネットワーク操作の方法

前節のような考え方でゲートを定義すると、3.5 節で仮定したゲートの動作すなわち「あるゲートの出力に要求があった場合、そのゲート自身が入力をチェックし、必要な入力が入力されていれば自分の処理をそれに加えて出力を計算する」という性質が各ゲートの機能として組み込まれる。本節では、プログラム上で、再帰的な方法あるいは総当たり法を実現する際の特徴、問題点をまとめ、本研究で用いる手法を決定する。

(1) 再帰法

再帰法においては、エンドノードに接続されたゲートから逆方向に順次ノードをたどってゆく。入力が不完全な場合は結果を先送りしてさらに前段のゲートで同様の処理を行うが、その間メンバー関数の呼び出し元の情報は、スタックと呼ばれる記憶領域上に積まれてゆくため、再帰的な定義が可能である。

ただしこの方法は、最大でエンド・ゲートからスタート・ゲートまでの最長ルートのゲート個数分だけスタックの領域が必要になるため、巨大なネットワークへ適用した場合には、予想外にデータ領域を消費することがあり、注意する必要がある。

(2) 総当たり法

総当たり法では、上記のクラスのメンバー変数としてゲートの状態(OPEN/CLOSE)を示すフラグを追加するとともに、派生クラスの GetOutput()の内容を次のように変更する。すなわち、処理に必要な前段のゲートが OPEN ならその出力を自身の入力として用いて出力を計算し、そうでなければ無視する。また、オブジェクトの生成時にはフラグの初期化を行う。ネットワークの処理は、フラグが CLOSE 状態にあるすべてのゲートに対して GetOutput()を実行する処理を繰り返し、すべてのゲートのフラグが OPEN になった時点で終了する。

この総当たり法では、スタートノードに直結したノードから順番に、ネットワークの後方に向かってノードが処理(OPEN)されてゆく。再帰的な処理に対してスタックは必要としないが、各ゲートにフラグのためのメンバー変数が必要となる。また、前段が OPEN 出ないゲートに対して何回もチェックがかかるため、演算が冗長になるという問題点がある。

再帰法はネットワーク全体を扱うプログラム・コードが簡潔に表現できるが、たとえばあるゲートの入力にそのゲート自身の出力が直接・間接に接続されるネットワーク構造の場合（フィードバックがある場合）、入力がいつまでも定まらず無限ループになる可能性がある。これに対して総当たり法では、このようにネットワーク構造にフィードバックがある場合でも、収束計算を行う道が残されているため、有利と考えられる。したがって、本研究においては総当たり法を選択する。

4.3 具体的なコード例

4.3.1 クラス宣言のコード

3.2 節で提案した情報クラス、ゲートクラスを具体的にコード化する例を紹介する。ただしここでは具体的な言語を C++とし、クラスの持つ属性・機能を列挙する。具体的なソース・コードについては参考1にまとめる。クラス全体の派生の関係は図 4.4 のとおりである。

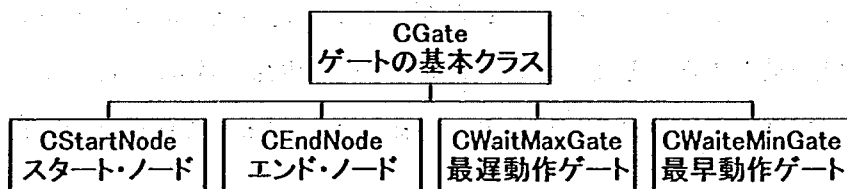


図 4.4 ゲートクラスのハイアラーキー

具体的なクラス構築の基本となる事項を列挙する。

- ・ 今回のケーススタディーでは、情報集約のための特別のゲートを定義はせず、エンド・ゲート

での出力を順次ファイルに書き込み、別のプログラムで整理した。このため情報集約ゲートは構築していない。

- ・ 最遅動作ゲートは、あらかじめ入力数に対する閾値すなわち制限時間値を持ち、その制限時間内に到達した入力のみが処理の対象となる。ただしこの値をゼロに設定すると、全ての入力を待つ。
- ・ ゲートの処理時間は固定値として設定する。実際のシミュレーションでは、後述するように正規乱数の結果をこれに設定している。

実際のコードについては参考 1 に表わし、以下では関係するクラスの概要について説明する。

(1) CInformation クラス

本クラスは、情報の到達時刻と情報量をメンバー変数に持つクラスであり、これらのメンバー変数は自由に参照可能である。時刻と情報量として double 型（実数）のメンバー変数のみを持ち、メンバー関数は定義していない。たとえば情報クラスの利用例は下記のようになる。

```
CInformation s1, s2;  
double inf;  
s1.time = 0;  
s1.info = 0;  
s2 = s1;  
inf = s2.info;
```

(2) CGate クラス

本クラスは、すべてのゲートクラスの基本となるクラスであり、複数の入力、情報の出力、ゲートの状態フラグといった必須のメンバー変数のほか、ゲートの名前などのメンバー変数を属性として有する。機能としては、ゲートの出力の計算、ゲートの接続といったすべてのゲートに共通なメンバー関数を宣言している。機能のうち全てのゲートに共通のものは具体的な動作が定義されているが、ゲートによって動作が異なるものは、その内容は定義されず、存在のみが"virtual"として宣言されている。たとえば入力数 2 のゲート A、B、C を生成し、それらを接続するには、次のように記述する。

```
CGate A, B, C;  
C.Connect( A );  
C.Connect( B );
```

(3) CStartNode クラス

このクラスは CGate クラスの派生クラスで、機能としてはトリガーとしてのみ意味を持つダミー情報を出力する。このゲートはネットワークのスタート・ノードとして用いられ、後続のゲートが動作を開始するきっかけをつくる。スタート・ノードなので入力数はゼロである。

このクラスは CGate クラスから派生しているが、実際に構築されるときにはコンストラクター CStartNode が呼び出され、入力数ゼロのゲートとを基本として生成される。また GetOutput は具体的には以下のように定義されており、常に時刻・情報量がゼロのダミー情報を出力する。

```
CStartNode::GetOutput()  
{  
    CInformation dmy;
```

```

    dmy.time = 0;
    dmy.info = 0;
    return dmy;
}

```

(4) CEndNode クラス

このクラスは機能的には CStartNode クラスと類似しているが、一つの出力と入力を持つ。

(5) CWaitMaxGate クラス

このクラスは、m_thresholdTime で指定された時刻の閾値以前に到着した全ての情報量の和を蓄積し、出力する。その動作は仮想関数として CGate で宣言された GetOutput 関数を具体的に実装することで実現している。このクラスの GetOutput は、入力側のすべてのゲートが OPEN になった時点で、入力側の情報のうち時刻が制限時間(m_thresholdTime)以前のものの和をとり、フラグを OPEN にセットするとともに出力情報 m_output を返す。入力ゲートのうち一つでも OPEN でなければそのまま初期値の m_output を返すが、その内容は不定である。

メンバー関数のうち、SetExecTime はこのゲートにおける処理時間をセットするほか、このゲートで発生する情報量 info を必要に応じてセットする。たとえば点検対象施設を表わすゲートとしてこれを利用する場合には、被災情報として info をセットする必要がある。

なお、このクラスは、入力情報を加工するだけでなく、被災施設の点検情報のように情報量を発生させるクラスとしても用いられる。

(6) CWaitMinGate クラス

このクラスは、CWaitMaxGate の反対の動作をするゲートで、入力情報のうち時刻が m_thresholdTime よりも早く、かつもっとも時刻が小さい情報を出力する。もちろん、この動作は入力ゲートのすべてが OPEN であるときのみ有効である。

4.3.2 クラスの操作法

前項で紹介したクラスを用いたネットワークを動作させる手順は以下のとおりである。

- ① 所要のオブジェクトの生成
- ② 処理時間や発生情報量など必要なゲート・パラメーターの設定
- ③ オブジェクトの接続によるネットワークの構築
- ④ 総当たり法によるネットワークの操作

特に④のネットワーク操作の流れは以下のとおりである。

- ① オブジェクト生成時にはすべてのゲートのフラグが CLOSE になっている。
- ② すべてのゲートをサーチしながらゲートの出力を計算するとともに、フラグの OPEN/CLOSE をチェック。
- ③ 全てのフラグが OPEN になるまで繰り返し。

このとき②においては、当該ゲートの `GetOutput` を呼び出すだけでそのゲートの種類に応じた `GetOutput` 関数が呼び出され、その関数内で当該ゲートの前段のゲートのフラグチェックが行われた後に出力が決定される。この部分のコードを、部分的に、実際に示してみよう。まず、いずれのゲートも `CGate` クラスから派生しているので、それを登録するための配列を宣言する。

```
CGate* p[CGATE_MAX]
```

この配列には、`CGate` から派生させた `CStartNode`、`CWaitMaxGate` など異なる種類のゲートを登録することができる。すべてのゲートが登録された状態で、以下の条件付き無限ループを実行する。

```
while(1) {  
    flg = OPEN;  
    for ( int i = 0; i < p_cnt; i++ ) {  
        p[i]->GetOutput();  
        if ( p[i]->GetOutputFlg() == CLOSE ) {  
            flg = 0;  
        }  
    }  
    if ( flg == OPEN ) break;  
}
```

このように、オブジェクト指向のアプローチによって主要部分のコードは非常に簡潔になる。

5. モデル体制によるケーススタディー

5.1 モデル体制の設定およびネットワーク表現

(1) 基本設定

提案した手法を用いて、具体的な工事事務所の体制を想定してパフォーマンスを評価する。モデルとなる工事事務所は国道を管理する事務所とし、実在の工事事務所のデータに基づき、以下のように諸元を設定した。

表 5.1 モデル事務所の諸元

	単位	数量
路線数	路線	4
1 路線あたり点検箇所数	箇所	50または10
出張所数	出張所	3
1 出張所あたり点検人員	人	5
地震計	台	5
ヘリコプター	機	1

また、想定する地震災害と体制を以下のように設定する。

- ・ 事務所管内を震源とする直下型地震が発生し、管内で最大震度 6 以上が観測される。
- ・ 地震によって発生する所管施設の被害の程度は、全施設のうち「被害大」が 20%、「被害小」が 30%、「被害なし」が 50%の割合とする。
- ・ 地震発生の時刻は勤務時間内（昼間）、勤務時間外（夜間）の 2 パターンを想定する。
- ・ 職員は昼間の場合出張所に、夜間の場合自宅にいるものとし、いずれの場合も地震発生を体感するものとする。
- ・ 夜間の場合、職員は自宅から最寄りの出張所まで参集した後点検活動を開始する。昼間の場合は各出張所から直接点検に出発する。なお、点検はあらかじめ定められたルートに沿って行われる。
- ・ 昼間の場合、あらかじめ定められた規則に基づきヘリコプターが飛行を開始し、上空から施設の被害を点検する。
- ・ 出張所数が 3、路線数が 4 のため、3 つのうち 1 つの出張所では 2 つのグループによって 2 路線を点検する。このとき人数の違いによる点検時間の差はないと考える。
- ・ 点検結果の報告は移動しながら行えるものとし、各施設の点検後に当該箇所の点検情報が事務所の窓口連絡されるものとする。

職員あるいはヘリコプターによる点検ルートを直感的に把握するため図 5.1、5.2 を示す。

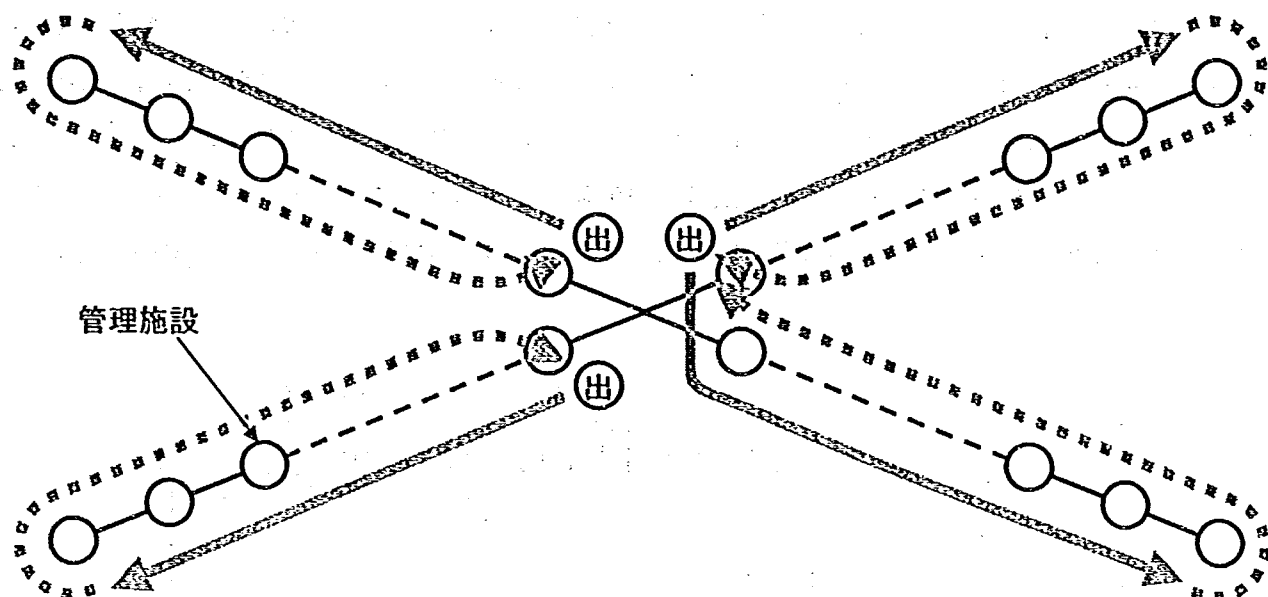


図 5.1 職員による地上点検のルート

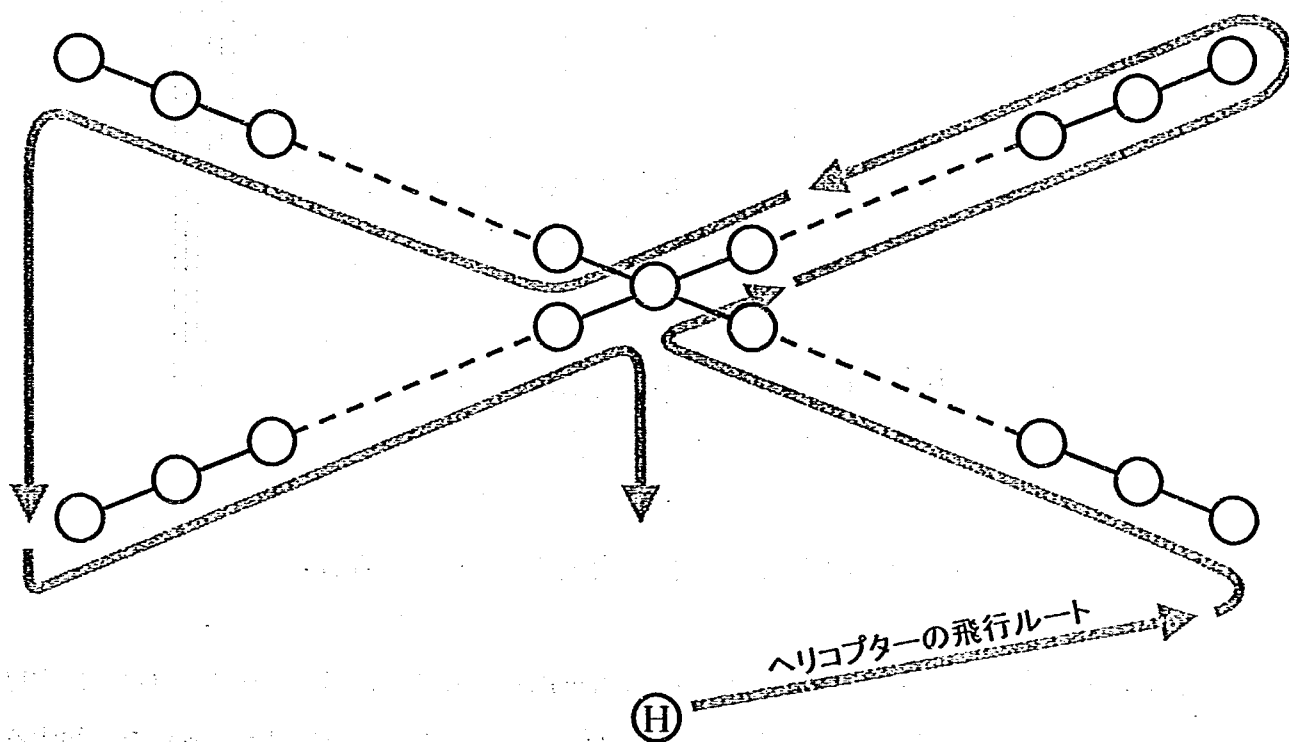


図 5.2 ヘリコプターによる空中からの点検ルート

(2) 体制のネットワーク表現

勤務時間内（昼間）の点検体制のネットワーク表現(DAY)を図 5.3 に示す。この場合は地震発生後、

5 台の地震計の情報が集約されるとともに、職員が体感により地震を感じ、報道等で地震状況を確認した後に点検活動を開始する。また同時にヘリコプターによる上空からの点検活動が開始される。職員による地上点検については、各ルート 50 箇所の施設に対して移動、点検を繰り返すこととしているが、後述するケース設定によってこの箇所数を 10 箇所とするケースもある。なお点検結果の報告は各施設の点検の後、次の箇所への移動と同時に行われ、その情報は情報集約ゲート(830)に集まる。

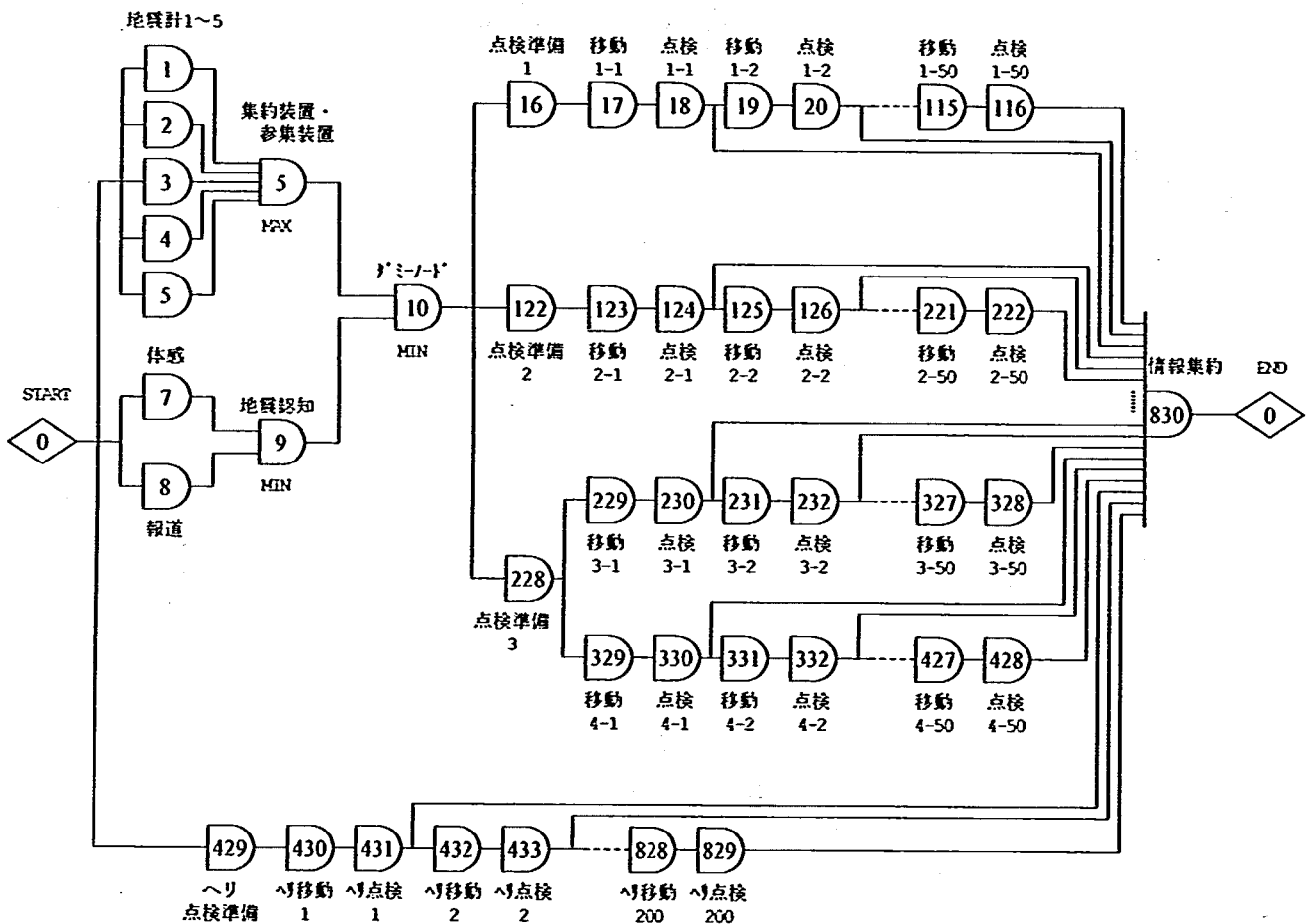


図 5.3 勤務時間内（昼間）の体制ネットワーク(DAY)

次に勤務時間外（夜間）の場合の体制ネットワーク(NIGHT)を図 5.4 に示す。ヘリコプターは有視界飛行のみ可能なため、この場合は上空からの点検を行えない。また、職員は自宅で地震を認知するか参集装置からの呼び出しを受けた後に出張所に参集し、点検活動を開始する。地上点検について、ここでは各出張所で計画要員がすべて参集した時点で（MAX ゲート）点検準備を経て点検活動が開始される。本来このような場合には、①一定の人員数になった時点で点検を開始する、②一人でも活動を開始し未点検の施設へ急行するといった体制の方がよいと考えられる。①の体制は点検準備のゲートとして「所定の数の入力ゲートが OPEN となった時点で参集ゲートを OPEN にする」という新

たなゲートクラスをつくれればよい。ただし、遅れて参集した要員を未点検の箇所へ向かわせるようなアルゴリズムが必要となる。同様に②については、状況に応じた点検ルートを組み替え（すなわち動的なネットワーク形態の変更）が必要になるため、アルゴリズムの高度化が必要となる。今回の試算では、被災箇所の分布などを変化させた複数ケースを評価するため、図 5.4 のネットワーク形態で統一した。なお、このケースについても、1 ルートあたりの箇所数に 50 箇所と 10 箇所の二つのパターンがある。

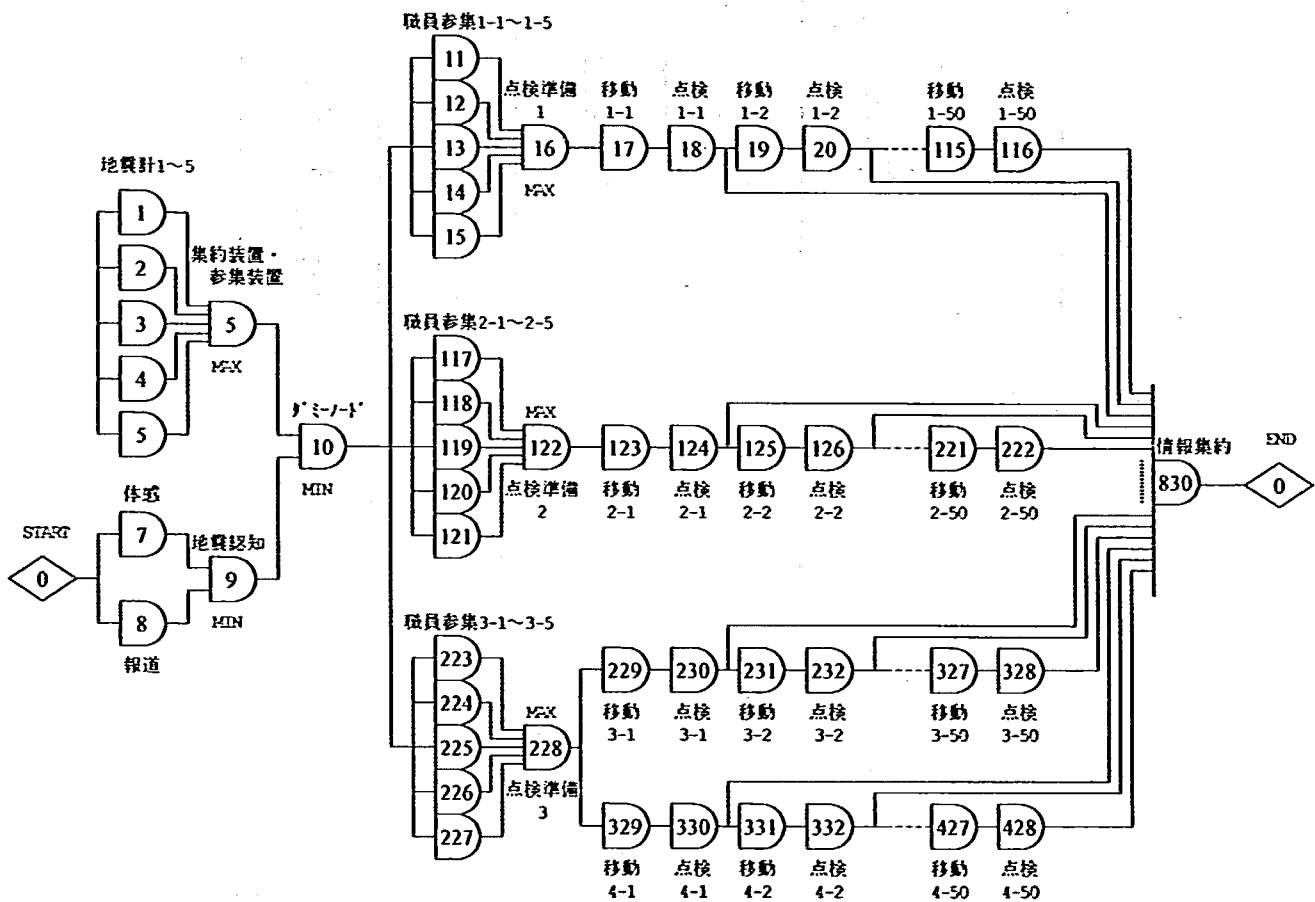


図 5.4 勤務時間外（夜間）の場合の体制ネットワーク

5.2 試算ケースの設定

5.2.1 ケース・パターンの設定

前節で DAY、NIGHT の二つのネットワークを設定したが、試算ケースとしては 1 ルートあたりの点検箇所数、被害分布等を変化させた複数のケースについて体制のパフォーマンスを試算する。そのヴァリエーションは以下のとおりである。

(1) 1ルートあたりの箇所数

50 箇所あるいは 10 箇所の二つのパターンを設定した。

(2) ルート間での被害分布

各ルートにおいて被害分布（20%、30%、50%）が同一のパターン（一様）と、4 つのルートで全体的な被害の程度に傾斜があるパターン（傾斜）。

表 5.2 ルート間の被害分布のパターン

	ルート	被害大	被害小	被害なし	合 計
一様	全体平均	20%	30%	50%	100%
	1	20%	30%	50%	100%
	2	20%	30%	50%	100%
	3	20%	30%	50%	100%
	4	20%	30%	50%	100%
傾斜	全体平均	20%	30%	50%	100%
	1	50%	50%	0%	100%
	2	20%	40%	40%	100%
	3	10%	20%	70%	100%
	4	0%	10%	90%	100%

(3) ルート内の被害程度の分布（あるいは点検の順番）

ルート内で被害程度にかかわらず点検を進めるか（ランダム）、被害が大きいものから先に点検するか（大→小）。本来ならば、このヴァリエーションは点検ルートの変更あるいは箇所間の移動時間の変更を必要とするが、今回の試算においては、ルート上の被害分布をランダムな順番にならべるパターンとルート内で被害大、小、なしの箇所を整列させるパターンとして設定した。

以上のパターンを組み合わせたケース設定の一覧を表 5.3 に示す。

表 5.3 ケース設定一覧

ケース	ヘリコプター点検	職員参集	1ルートあたり箇所数	ルート間被害分布	ルート内被害分布
DAY-1	可能	不要	50	一様	ランダム
DAY-2	可能	不要	50	傾斜	ランダム
DAY-3	可能	不要	50	一様	大→小
DAY-4	可能	不要	10	一様	ランダム
DAY-5	可能	不要	10	傾斜	ランダム
DAY-6	可能	不要	10	一様	大→小
NIGHT-1	不可能	必要	50	一様	ランダム
NIGHT-2	不可能	必要	50	傾斜	ランダム
NIGHT-3	不可能	必要	50	一様	大→小
NIGHT-4	不可能	必要	10	一様	ランダム
NIGHT-5	不可能	必要	10	傾斜	ランダム
NIGHT-6	不可能	必要	10	一様	大→小

また、各ケースのルートごとの箇所数を表 5.4 に示す。

表 5.4 ルートごとの被害程度別箇所数

ケース	ルート	被害大	被害小	被害なし	合 計	備 考
DAY-1	全ルート	10	15	25	50	ランダム
DAY-2	ルート1	25	25	0	50	
	ルート2	10	20	20	50	
	ルート3	5	10	35	50	
	ルート4	0	5	45	50	
DAY-3	全ルート	10	15	25	50	大→小
DAY-4	全ルート	2	3	5	10	ランダム
DAY-5	ルート1	5	5	0	10	
	ルート2	2	4	4	10	
	ルート3	1	2	7	10	
	ルート4	0	1	9	10	
DAY-6	全ルート	2	3	5	10	大→小
NIGHT-1	全ルート	10	15	25	50	ランダム
NIGHT-2	ルート1	25	25	0	50	
	ルート2	10	20	20	50	
	ルート3	5	10	35	50	
	ルート4	0	5	45	50	
NIGHT-3	全ルート	10	15	25	50	大→小
NIGHT-4	全ルート	2	3	5	10	ランダム
NIGHT-5	ルート1	5	5	0	10	
	ルート2	2	4	4	10	
	ルート3	1	2	7	10	
	ルート4	0	1	9	10	
NIGHT-6	全ルート	2	3	5	10	大→小

5.2.2 パラメーター設定

(1) 被災箇所の情報量

3.1 で定義した情報量の考え方にに基づき、各点検箇所の被害程度と点検方法に対して得られる情報量の値を表 5.5 のように定義する。

表 5.5 被害程度と点検方法に対する情報量

	被害大	被害小	被害なし
地上点検	1.0	1.0	1.0
ヘリ点検	1.0	0.5	0.5

(2) 各ゲートの処理時間

現在のクラス設計においては、各ゲートの処理時間は定数で与えることになっている。一方、実際の体制においては、多くの処理等に要する時間は変動する。そこで試算においては、各ゲートのうち処理時間を変動させるものについては、時間を設定する際に乱数を発生させ、その値をゲートの処理時間として設定した。乱数は正規分布に従うとし、ゲートの種類ごとに平均値と分散を設定した（表 5.6）。

特に点検時間については、被害程度が大きいほど長い時間を要することとしたほか、夜間の方が昼

間より点検時間が長いと設定した。

表 5.6 ゲートの処理時間の設定

(a) 処理時間を一定とするもの（昼間・夜間共通）

ゲートの種類	処理時間	備 考
START	0.0	
集約装置・乱数装置	5.0	
体 感	3.0	
報 道	5.0	
地震認知	0.0	
ダミーノード	0.0	
報 告	0.5	
ヘリ点検準備	60.0	
ヘリ移動 1	30.0	ヘリポート～最初の点検箇所
ヘリ移動（1以外）	1.0	点検箇所間
END	0.0	

(b) 処理時間を変動させるもの（点検活動以外）

	昼 間		夜 間	
	平均	標準偏差	平均	標準偏差
地震計	2.0	0.5	2.0	0.5
職員参集	-	-	50.0	20.0
点検準備	30.0	10.0	30.0	10.0
移 動	5.0	2.0	7.0	3.0
情報集約	20.0	5.0	20.0	5.0

(c) 処理時間を変動させるもの（点検活動）

	昼 間		夜 間	
	平均	標準偏差	平均	標準偏差
地上点検・被害大	15.0	5.0	20.0	7.0
地上点検・被害小	10.0	3.0	13.0	5.0
地上点検・被害なし	7.0	2.0	10.0	3.0
ヘリ点検・被害大	3.0	1.0	-	-
ヘリ点検・被害小	2.0	1.0	-	-
ヘリ点検・被害なし	1.0	0.5	-	-

※単位はすべて [分]

5.3 結果

昼間 6 ケース、夜間 6 ケースのネットワーク形態および計算結果を附図 1～12 に掲載する。本文では、これらの結果を集約した図によって計算結果に対して考察を加え、手法の妥当性を検討する。

(1) 昼間と夜間の比較

まず、点検箇所数が 50 箇所の場合について、昼間のケース (DAY-1～3) および夜間のケース (NIGHT-1～3) の情報量蓄積状況を図 5.5、5.6 に示す。いずれの図についても、地震発生時点（時刻 0）から点検情報が窓口に集まり始めるまで（すなわち累積曲線が立ち上がるまで）ある程度のタイムラグを要していることがわかる。また累積情報量は、全箇所 50 箇所×4 ルート×1.0=200 が最大であり、全ての箇所の被災内容が判明した場合にこの値となる。

まず両図の傾向を比較すると、情報収集開始の時点、情報収集の速さ（累積曲線の傾き）、情報収

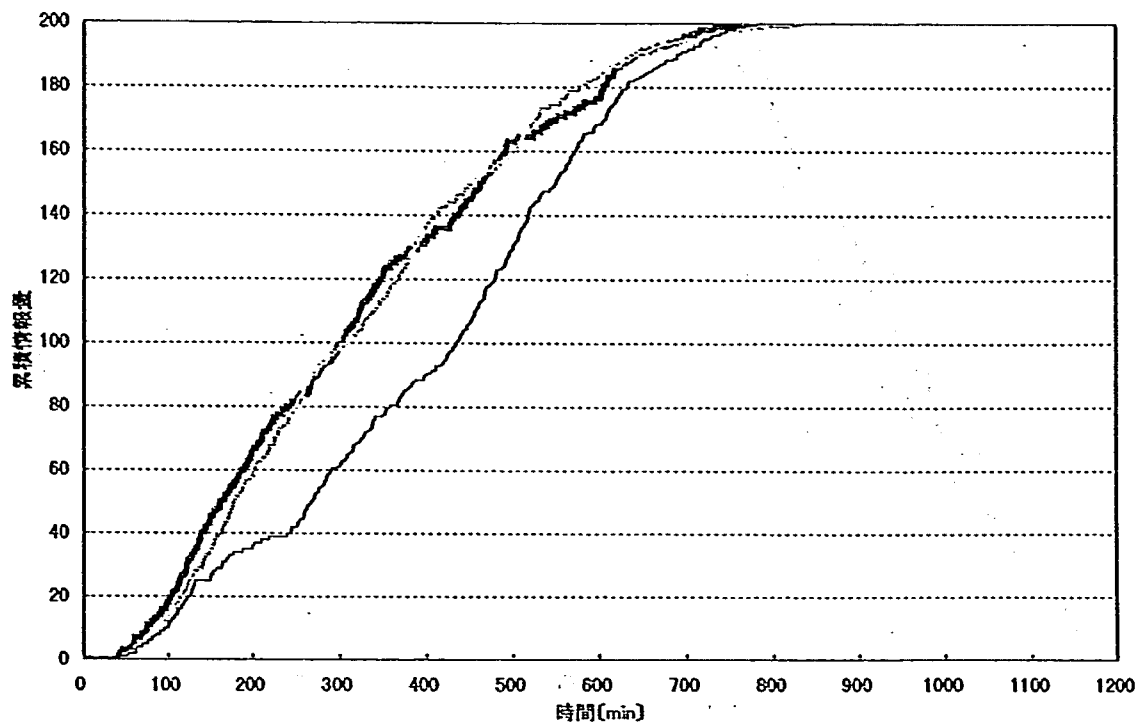


図 5.5 情報量蓄積状況の比較（箇所数：50 箇所／ルート、昼間）

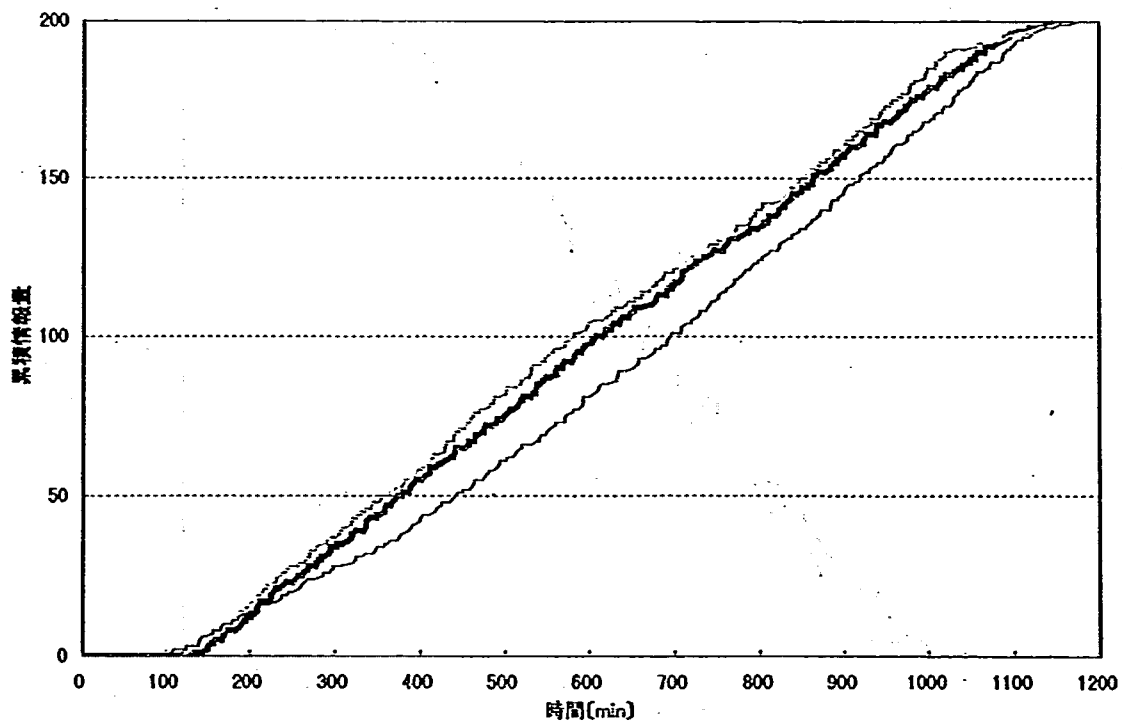


図 5.6 情報量蓄積状況の比較（箇所数：50 箇所／ルート、夜間）

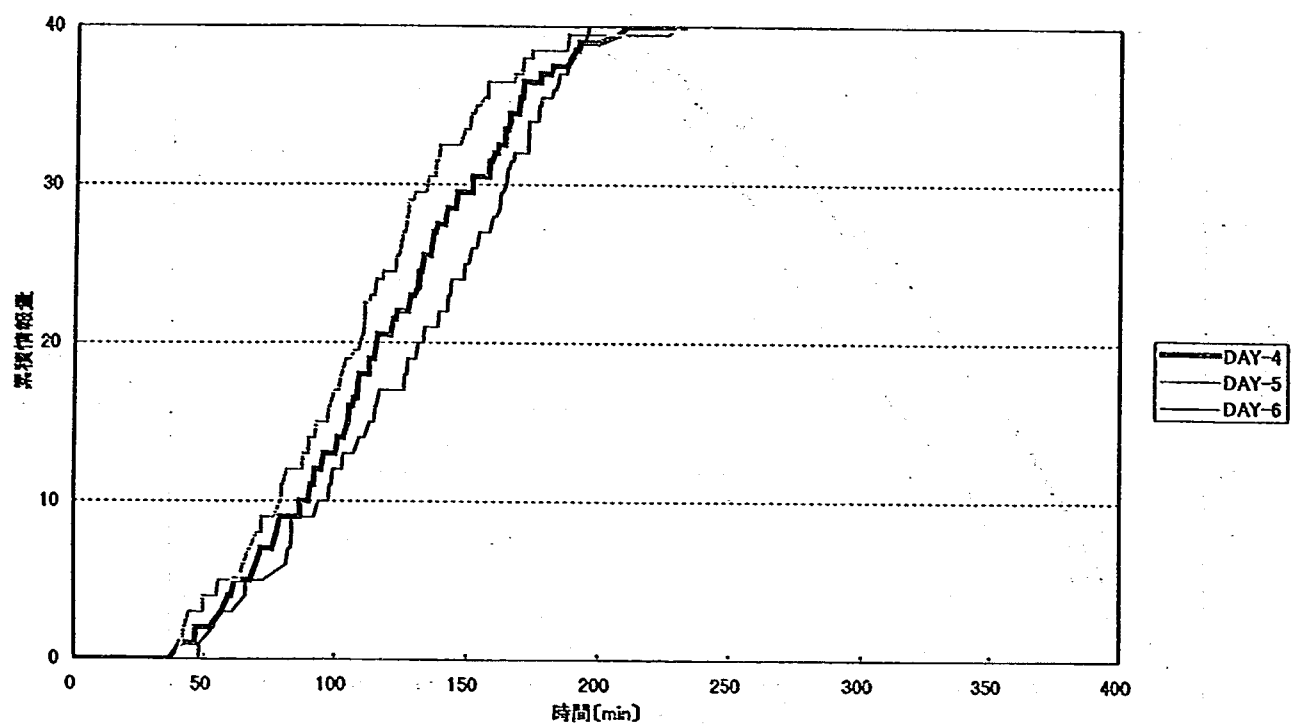


図 5.7 情報量蓄積状況の比較（箇所数：10 箇所／ルート、昼間）

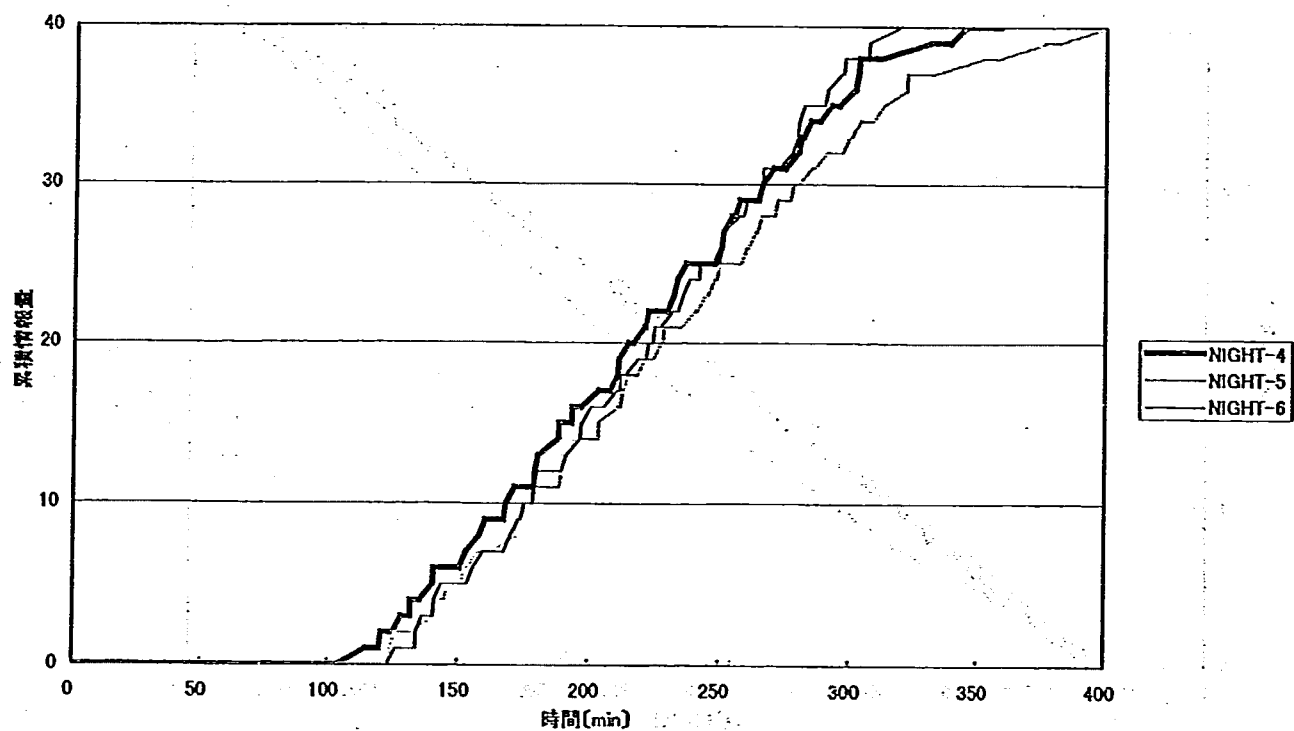


図 5.8 情報量蓄積状況の比較（箇所数：10 箇所／ルート、夜間）

集完了時刻いずれについても、昼間の方が早いことがわかる。情報収集の迅速性については、昼間の方が夜間より点検時間が短いと設定しているため当然であるが、このほか、昼間の DAY-1、DAY-2 については、ヘリコプターがいち早く上空からの点検を開始するため立ち上がりが高く、また傾きも上に凸となっている。震災対策体制について本研究で提案する手法を用いることで、このように定量的なパフォーマンス評価が可能となる。

(2) 被害程度と点検の順番

DAY-3 については累積曲線が下に凸になっているが、これは被害大、被害小、被害なしの箇所の順番に点検を行うためである。すなわち、立ち上がりではヘリコプターによって被害大の点検情報(情報量 1.0) が迅速に収集されるが、その後いったん地上点検で時間を要するため情報収集が遅くなり、被害規模が小さな箇所に移るに連れて再び情報収集速度が速くなる。このことは、DAY-3 の曲線に、当所の急峻な立ち上がりおよび被害大、小、なしに対応したわずかな段差が見られることからわかる。一方夜間については、地上点検のみが行われるため、累積曲線が全体的に直線に近い。ただしここでも、被害が大きいものから点検する NIGHT-3 のケースでは、被害大を点検する最初は情報収集速度が遅い。

以上の結果だけを見ると、被害程度が大きいところから点検するケースは、全体的な情報収集速度が遅く不利に見えるが、この場合は先に被害程度が大きい箇所の情報が確実に得られていることに留意する必要がある。したがって、情報量の概念に被害程度を含めた考え方を導入した方が、体制のパフォーマンスをよりの確に表現できる。

(3) ルート間で被害程度が異なる場合

また被害程度の分布が全ルートで同じケース(DAY-1、NIGHT-1)とルート間で異なるケース(DAY-2、NIGHT-2)では曲線に大きな違いはない。ここで、たとえば DAY-1 と DAY-2 のケースについてかくルートごとの点検所用時間を比較してみる(附図 1(c)および附図 2(c))。全てのルートで被害程度の分布が一樣な DAY-1(附図 1(c))の場合、確率的変動によって若干の差はあるものの、四つの地上点検ルート間で点検所用時間に大きな差はない。一方、ルート間で被害程度の分布が異なる DAY-2(附図 2(c))の場合、被害大の箇所が多い地上点検ルート 1 の所用時間が最も長く、ルート上の被害程度の分布と点検所用時間が対応している。もしここで情報量に被害程度の概念が含まれていれば、各ルート同じ人員で点検活動を行う体制では被害程度が大きいルートの(すなわち価値が高い情報を多く持つルートの)情報収集時間が長くなるはずである。これに対して被害が大きいルートに人員を多く配置するようにすれば、より効率的に点検活動を行えることが計算結果にも表れると予想される。

なお図 5.7 および図 5.8 は、箇所数が 1 ルートあたり 10 箇所のケースについて情報累積状況を示したものである。全体的な傾向は先の多いケースと同じであるが、箇所数が多いケースよりも各ケース間の差が小さい。

(4) 総合的な点検所用時間

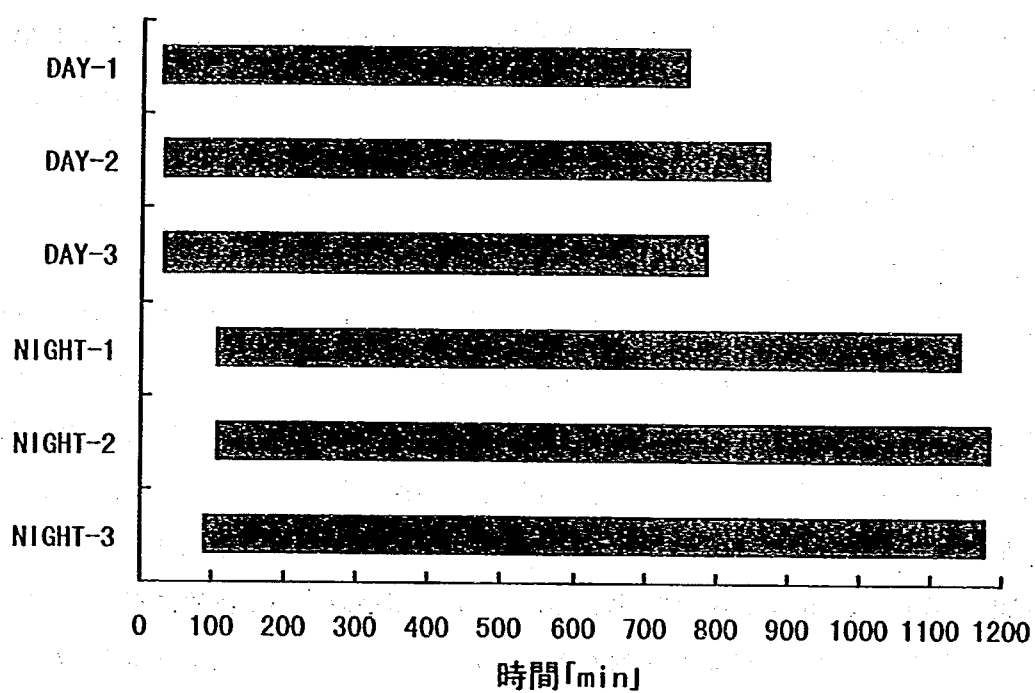


図 5.9 点検所用時間の比較（箇所数：50 箇所／ルート）

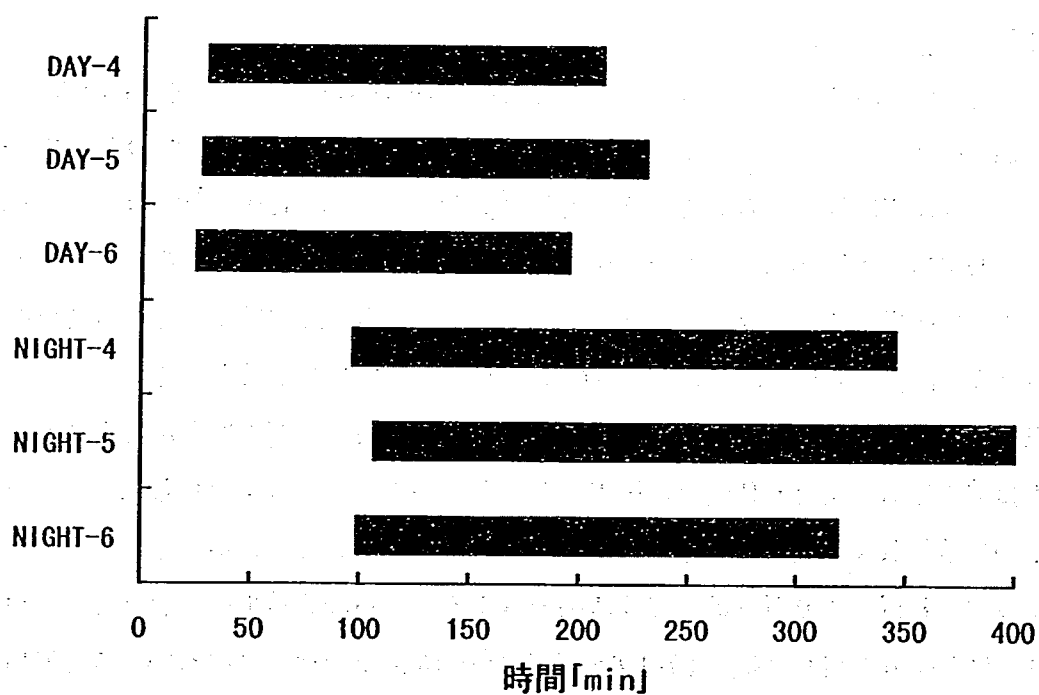


図 5.10 点検所用時間の比較（箇所数：10 箇所／ルート）

地上点検のルート、ヘリコプターによる上空からの点検を問わず、点検開始から全ての箇所情報が確定するまでの時間を図示したのが図 5.9、5.10 である。箇所数が多い場合、少ない場合のいずれにおいても、昼間の方が点検活動が迅速に行われることが表わされている。また、すべての場合でルート間に被害程度の差があるケース（DAY/NIGHT-2、5）の所用時間が長くなっており、事前に固定された体制の非効率性が表わされている。たとえばこのような場合、地震動の即時モニタリングや施設被害の即時推定に基づいて点検体制を変更することにより、より効率的な点検活動が可能となる。

6. 結 論

本研究では、人、機器からなる震災対策体制を抽象的なネットワークとして表現することにより、体制の迅速性のパフォーマンスを定量的に評価する手法を提案した。そしてその手法を、モデルとして設定した建設省工事事務所に対して適用し、その妥当性を検証した。その結果、本手法によってこれまで定性的にのみ把握されてきた（あるいはまったく意識されていなかった）体制のパフォーマンス、ボトルネックを評価しうることがわかった。しかしながら、今回提案された手法においては、以下のような課題が認識された。

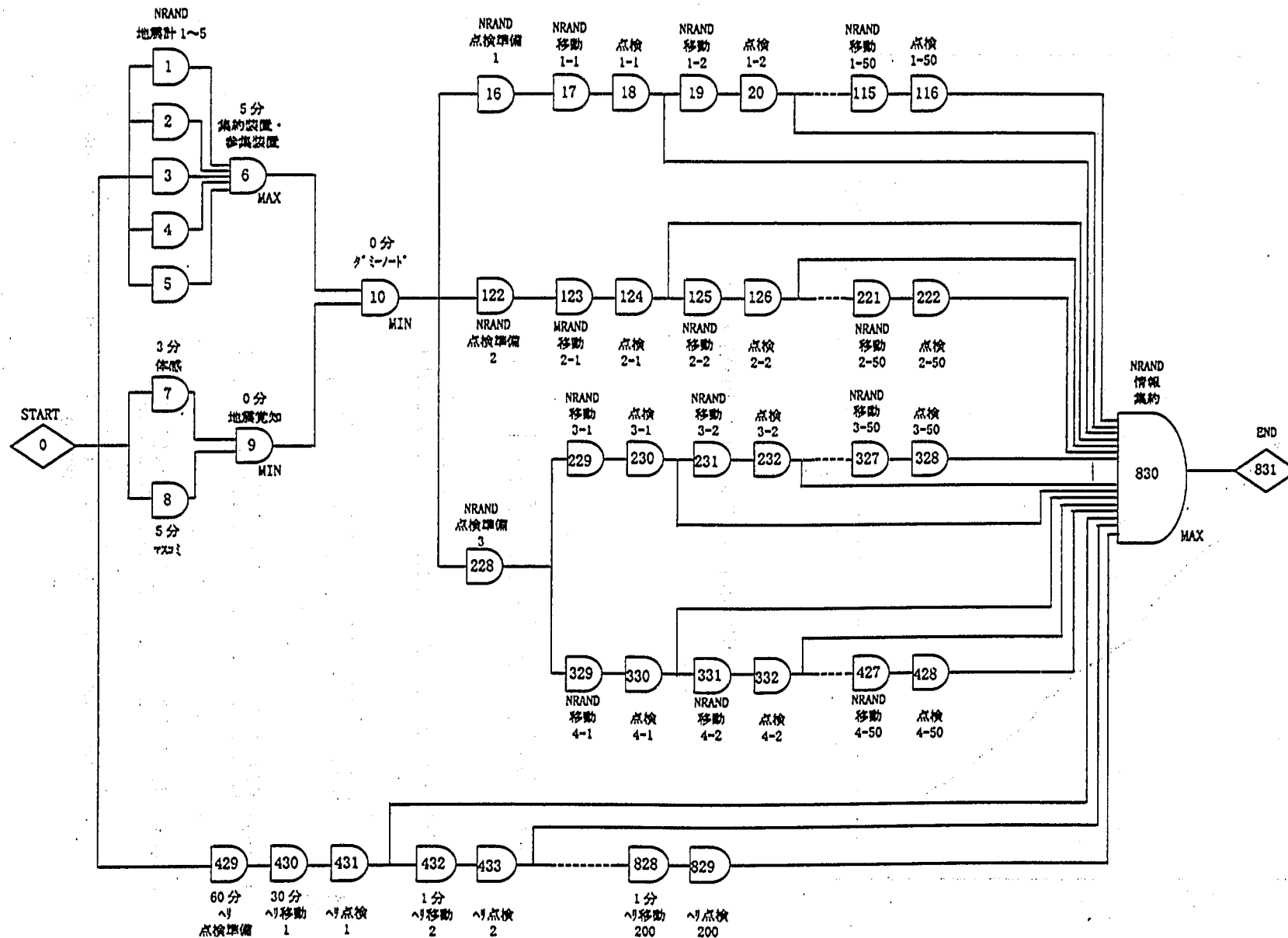
- (1) 今回の情報量の定義では施設被害の程度は含まれず、その確からしさのみを指標としたが、震災対策体制においては被害程度が大きい箇所の情報が重要となる。たとえば現在の情報量の定義でも、被害程度ごとにこれを集計し、定量化することができるが、被害程度自体を組み入れた情報量の定義を検討する必要もある。
- (2) 参集不能、機器の故障といった点検リスクを組み入れる必要がある。そのためには、実際の体制におけるそのようなリスクの計測法を開発する必要がある。また同様に、ネットワーク中の各ゲートの処理時間を計測・同定する方法も検討する必要がある。
- (3) ネットワーク設定時の体制を被害状況に応じて柔軟に変化させることができないため、より高度に計画された震災対策体制を表わすことができない。ネットワーク形態を柔軟に変化させることができるアルゴリズムを考える必要がある。

本手法を具体化する際に用いたオブジェクト指向によるアプローチは、与えられた機能を持ったオブジェクトを任意に生成し、その挙動は各オブジェクトに任せることができるため、クラス設計次第で様々な動作をする人、機器等のオブジェクトを生成することが可能である。したがって本手法の考え方は、複雑なシステムを簡単な要素の組み合わせで表現するという意味で、非常に大きな可能性を持つといえよう。

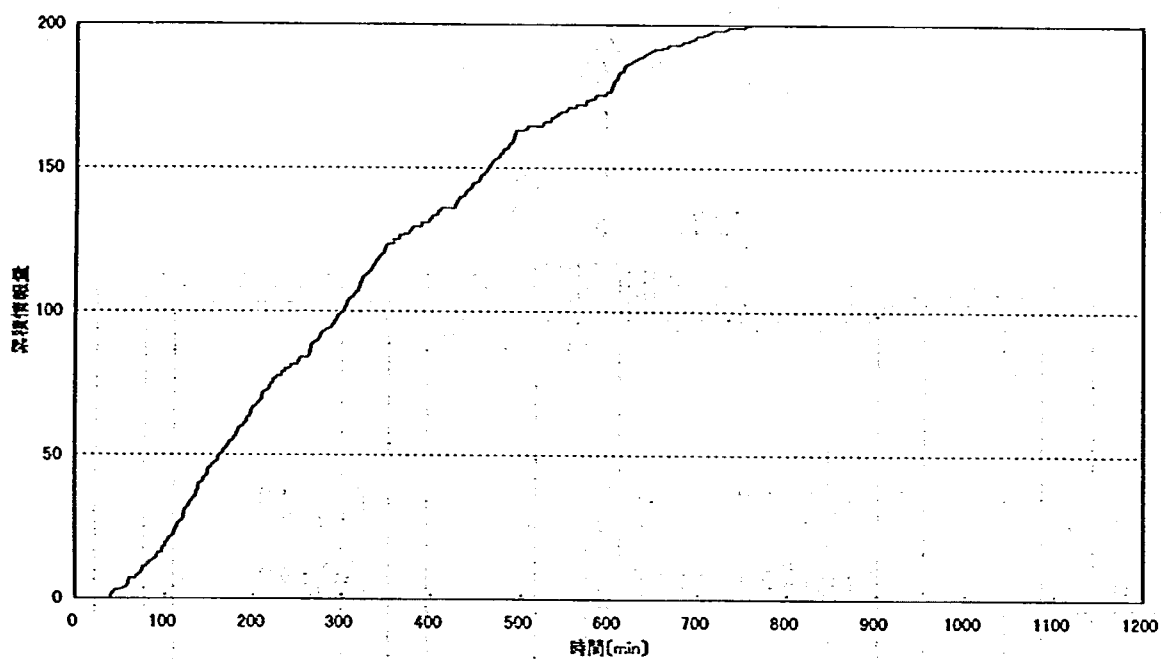
参考文献

- 1) 中央防災会議, 「防災基本計画」, 1997 年 6 月
- 2) 「建設省防災業務計画」, 1998 年 3 月
- 3) たとえば (社) 日本道路協会, 「道路震災対策便覧 (震後対策編)」, 1996 年 10 月
- 4) 建設省総合技術開発プロジェクト「災害情報システムガイドライン (案)」, 1992 年 3 月
- 5) 山崎文雄, 「リアルタイム地震防災システムの現状と展望」, 土木学会論文集 No.577, 1997 年 10 月, pp1
- 6) 金子正洋, 杉田秀樹, 濱田禎, 「建設省における震後即時被害予測システムの開発」, 土木学会第 52 回年次学術講演会講演概要集 (1), 1997 年 9 月, pp816
- 7) 片田敏孝, 青島縮次郎, 及川康, 「災害情報システム評価のためのシミュレーションモデルの開発」, 土木計画学研究・講演集 No.19(1), 1996 年 11 月, pp21
- 8) 木俣昇, 鷺見育男, 「消防防災システムの阻害要因のペトリネットシミュレーション」, 土木計画学研究・講演集 No.19(2), 1996 年 11 月, pp43
- 9) 野崎智文, 杉田秀樹, 遠藤和重, 「都市基盤施設の被害見地体制に関する研究」, 第 24 回地震工学研究発表会講演論文集, 1997 年 7 月, pp1273

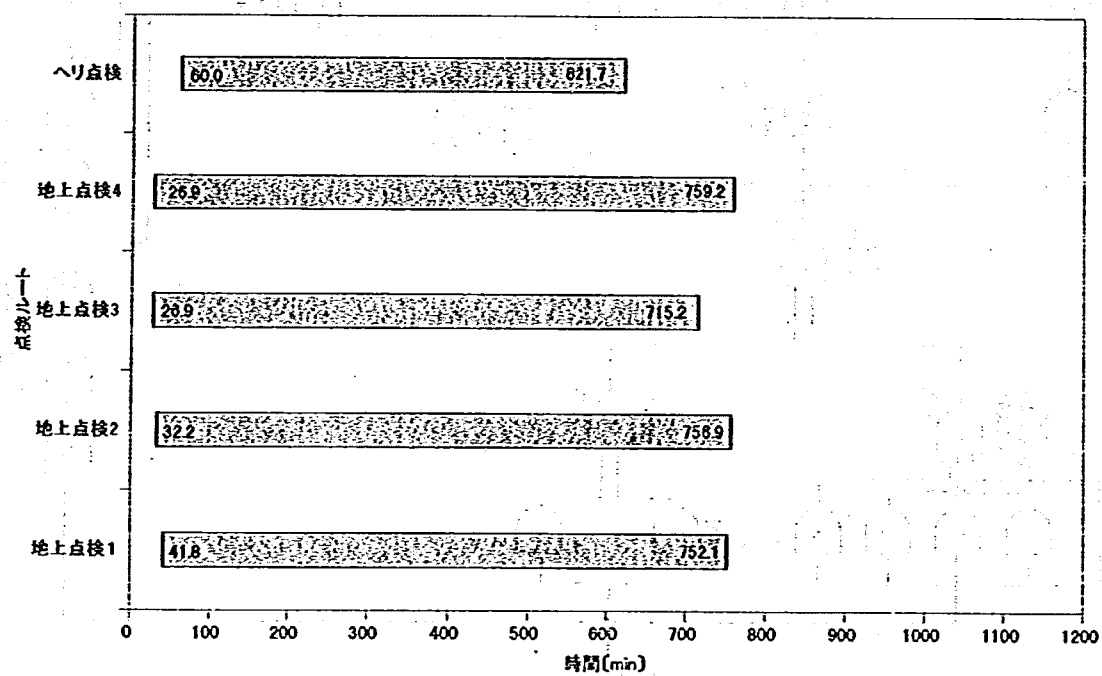
附 圖



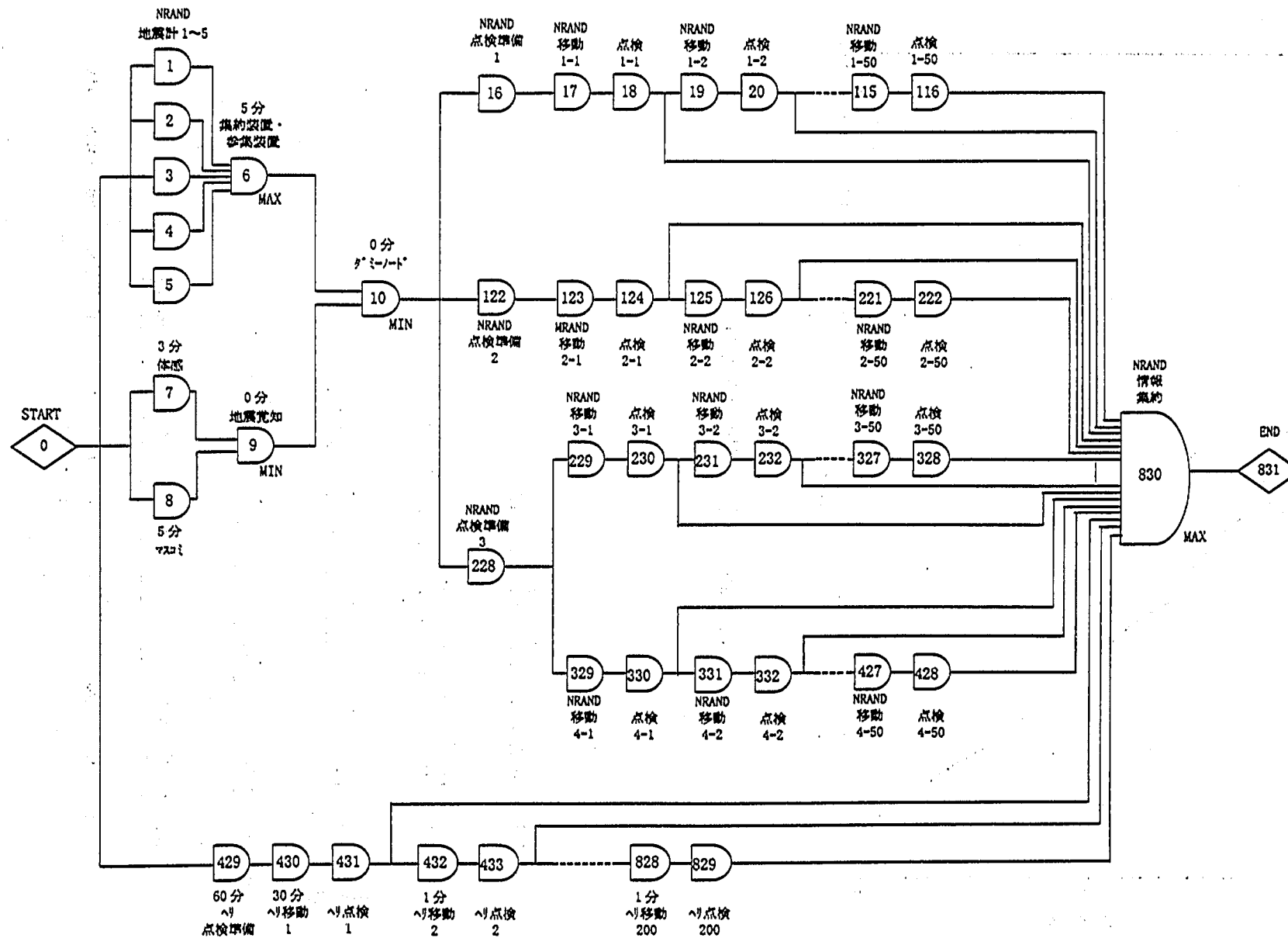
附図 1(a) モデルネットワーク (DAY-1)



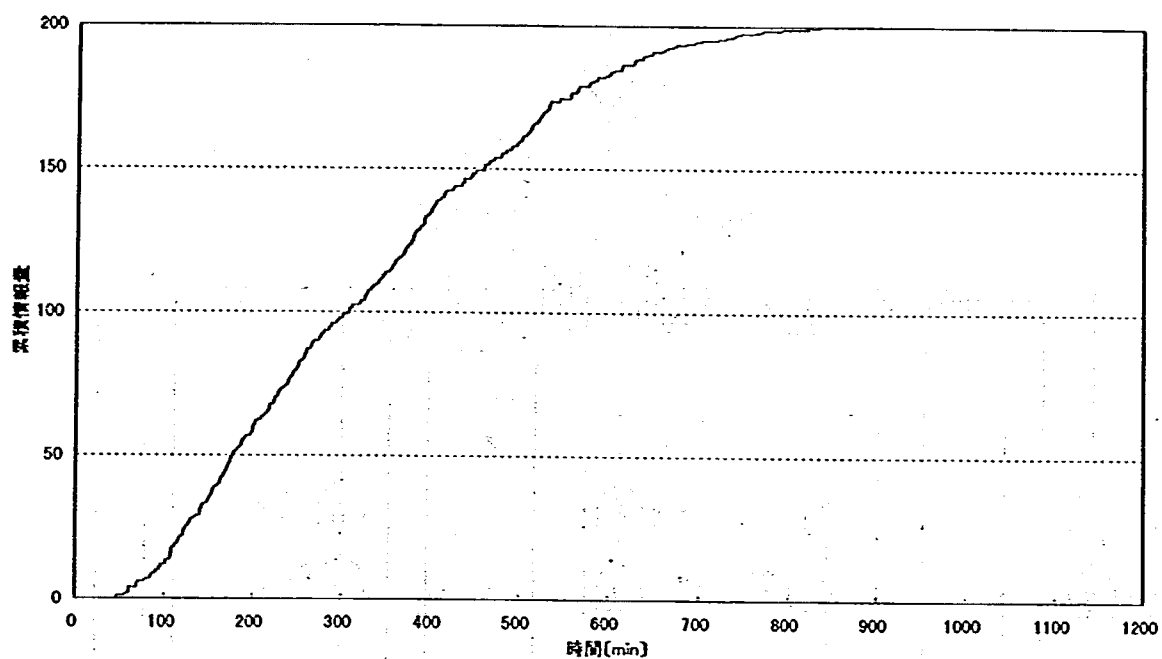
附図 1(b) 情報量累積状況(DAY-1)



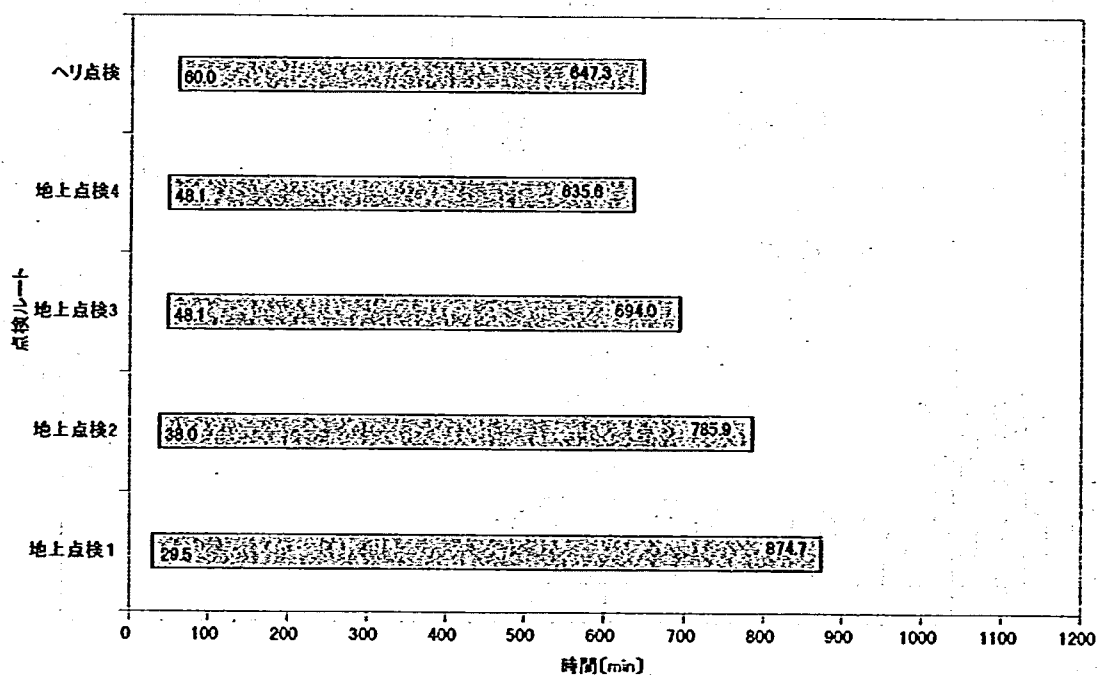
附図 1(c) 点検所用時間(DAY-1)



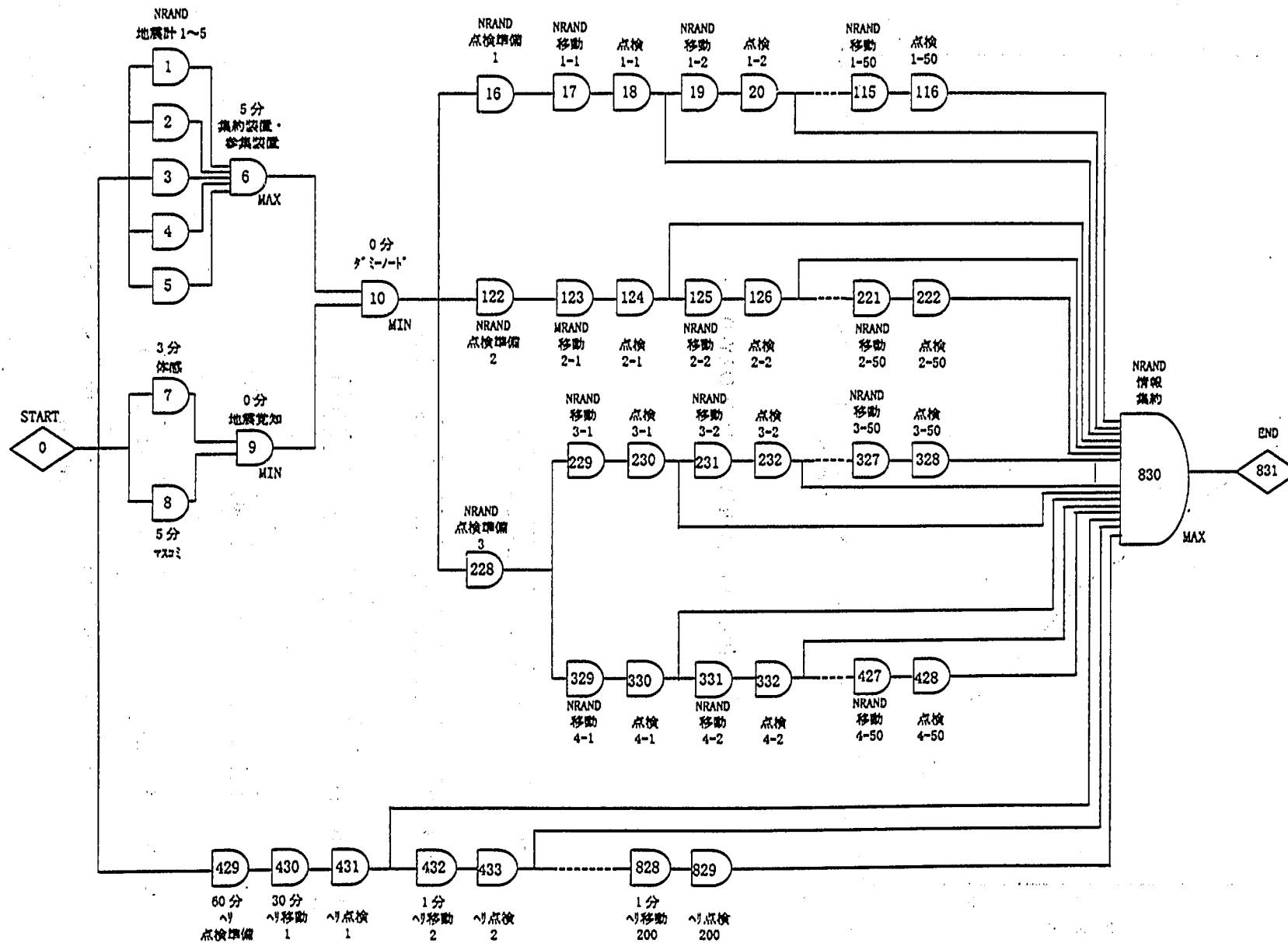
附図2(a) モデルネットワーク (DAY-2)



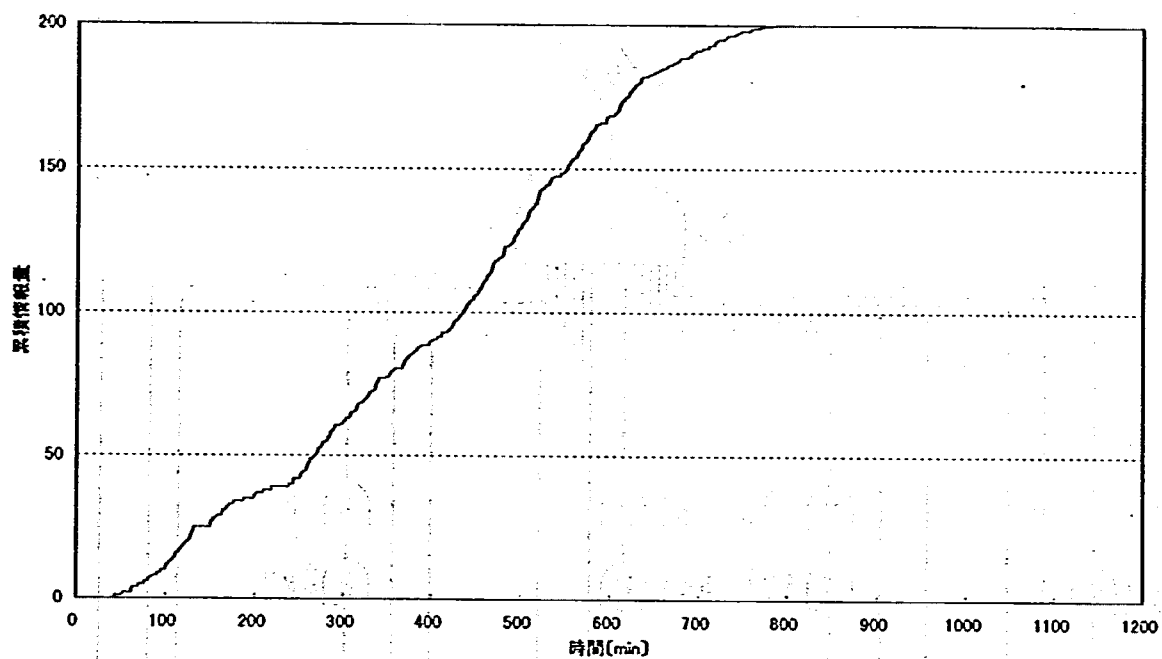
附図 2(b) 情報累積状況(DAY-2)



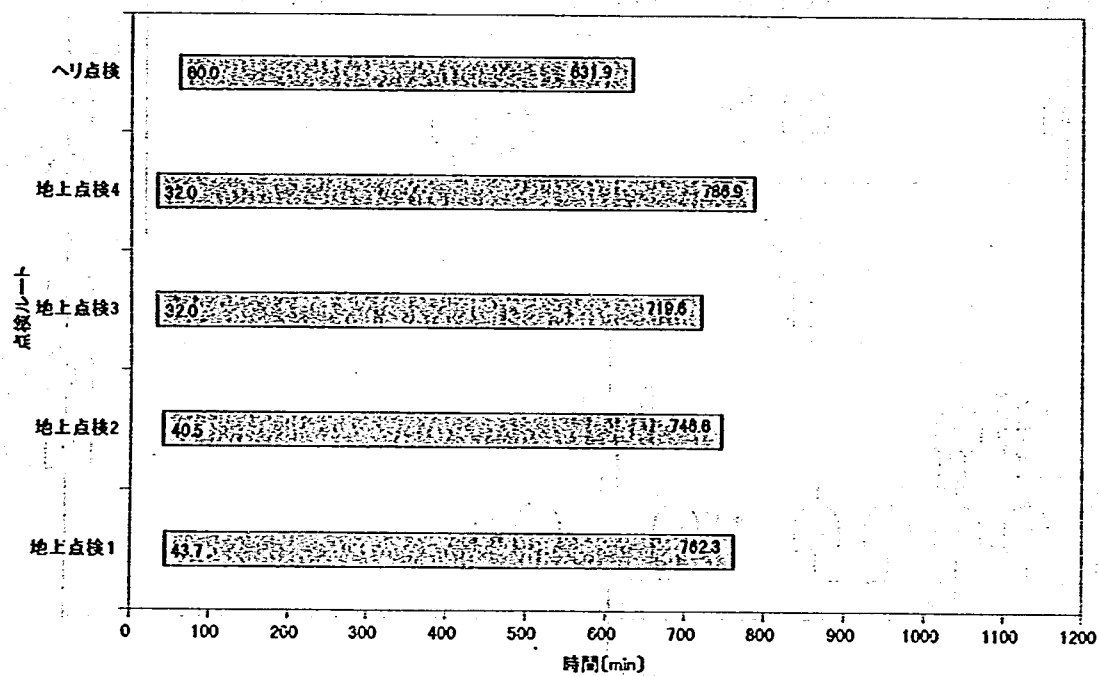
附図 2(c) 点検所用時間(DAY-2)



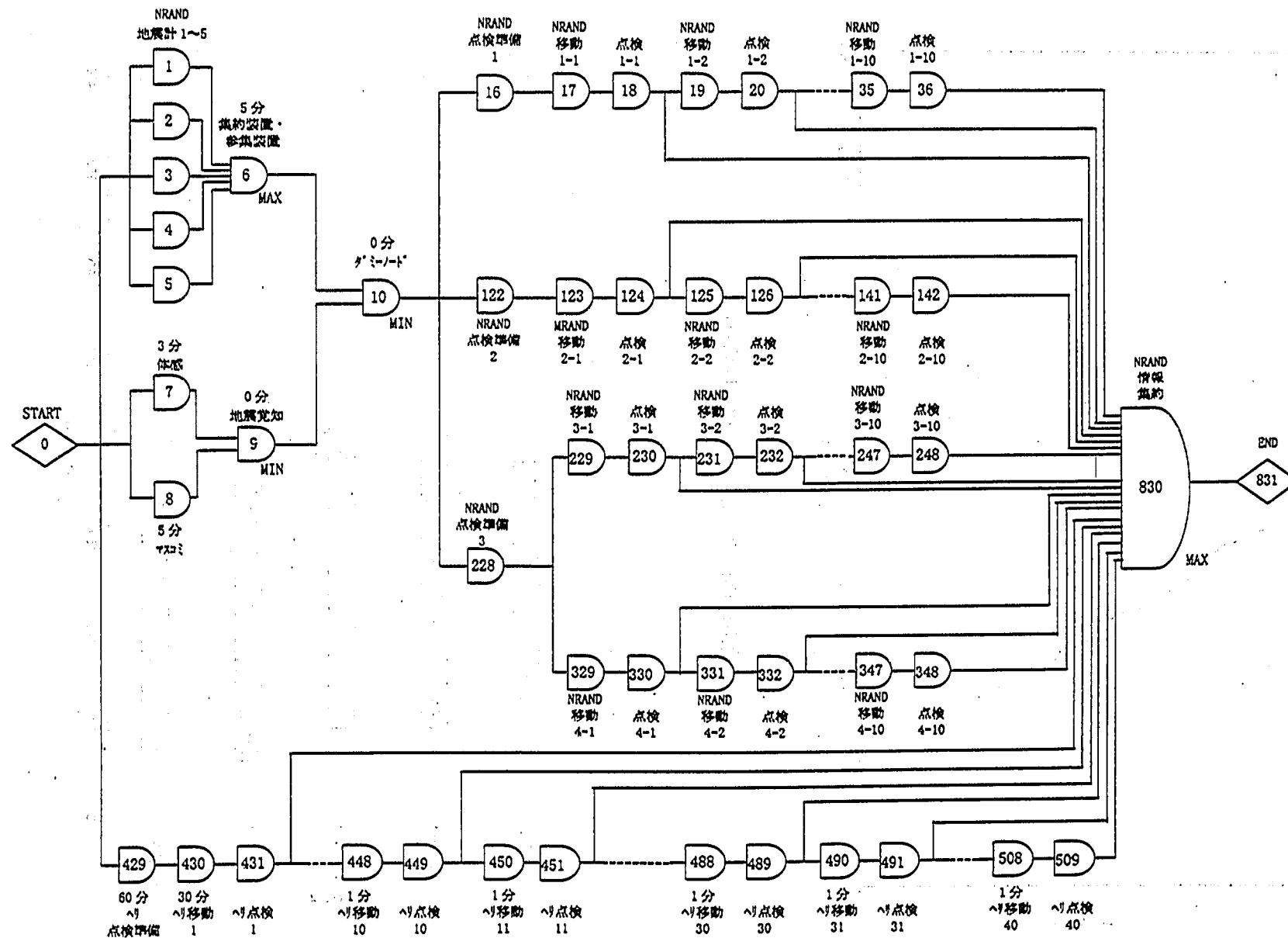
附図 3(a) モデルネットワーク (DAY-3)



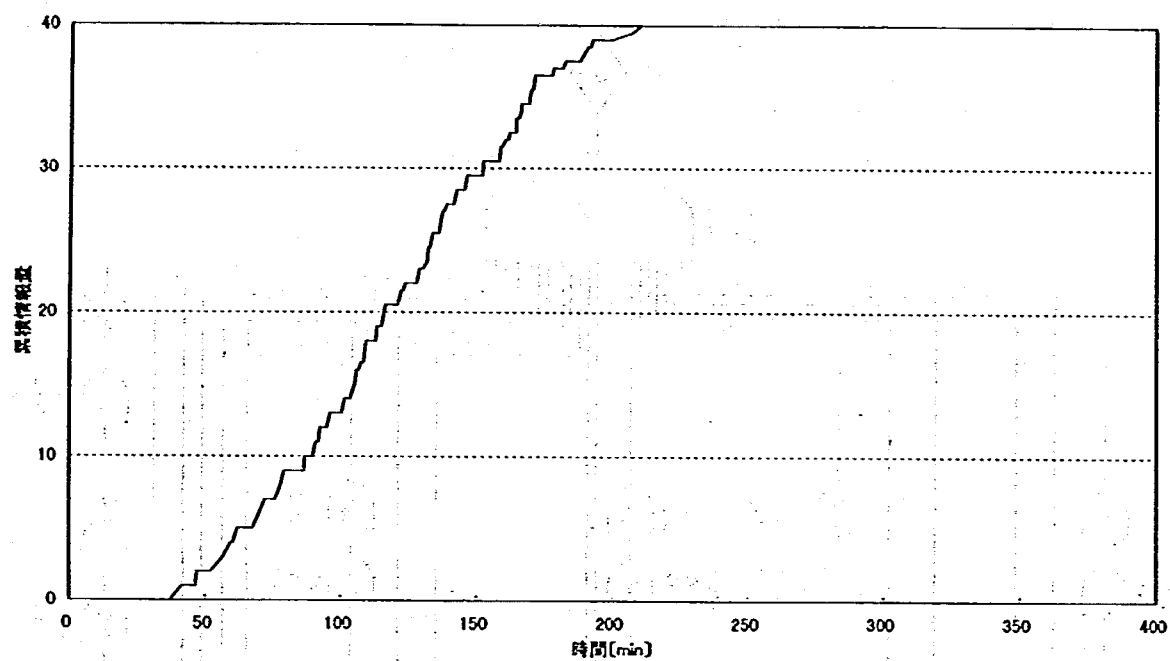
附図 3(b) 情報量累積状況(DAY-3)



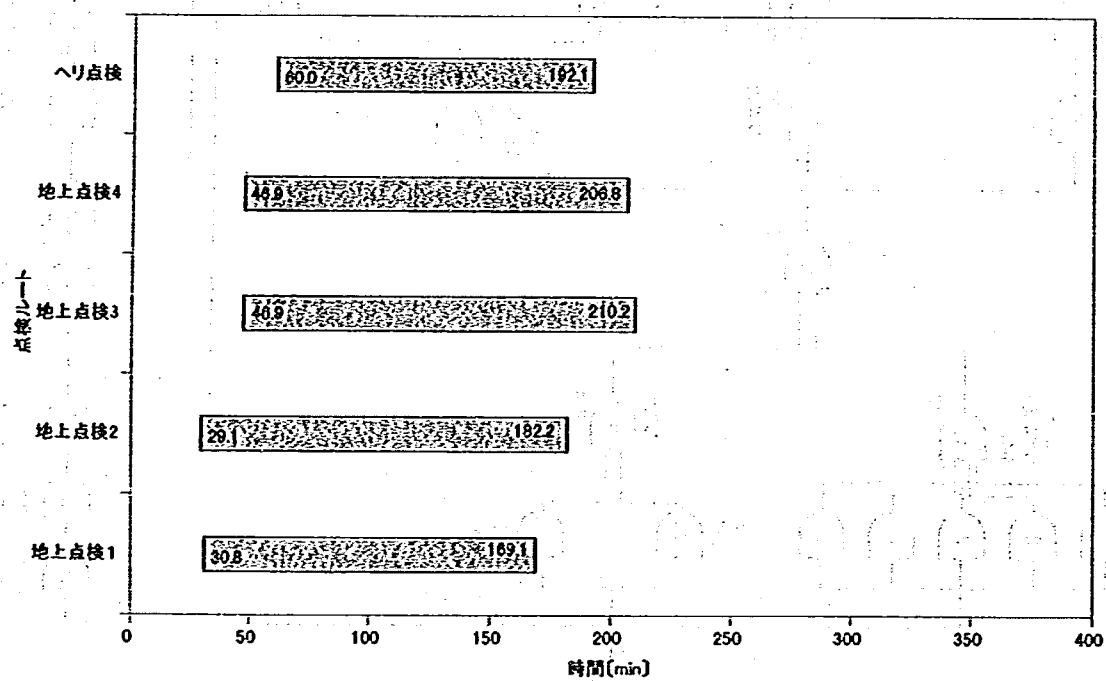
附図 3(c) 点検所用時間(DAY-3)



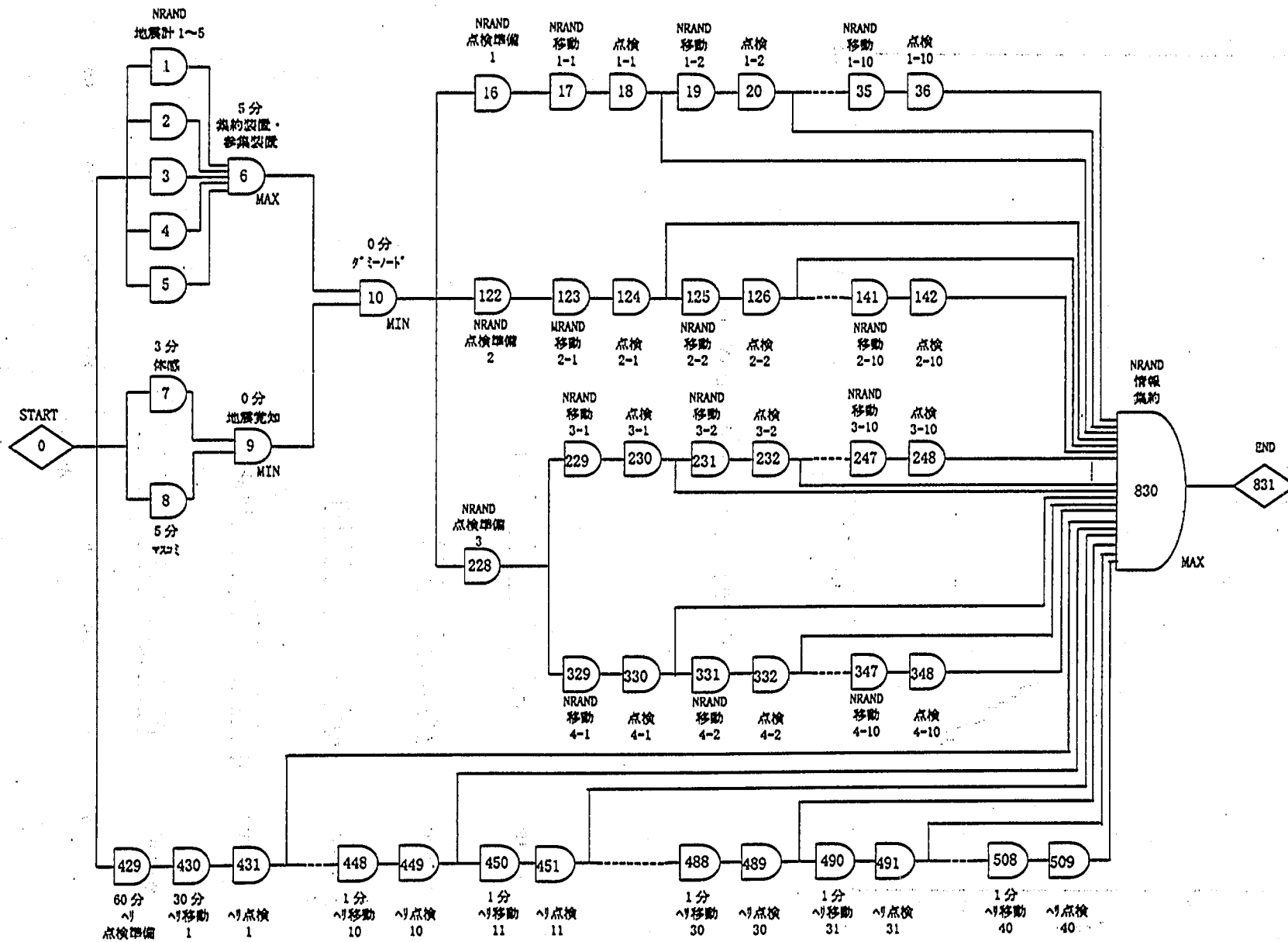
附図 4(a) モデルネットワーク (DAY-4)



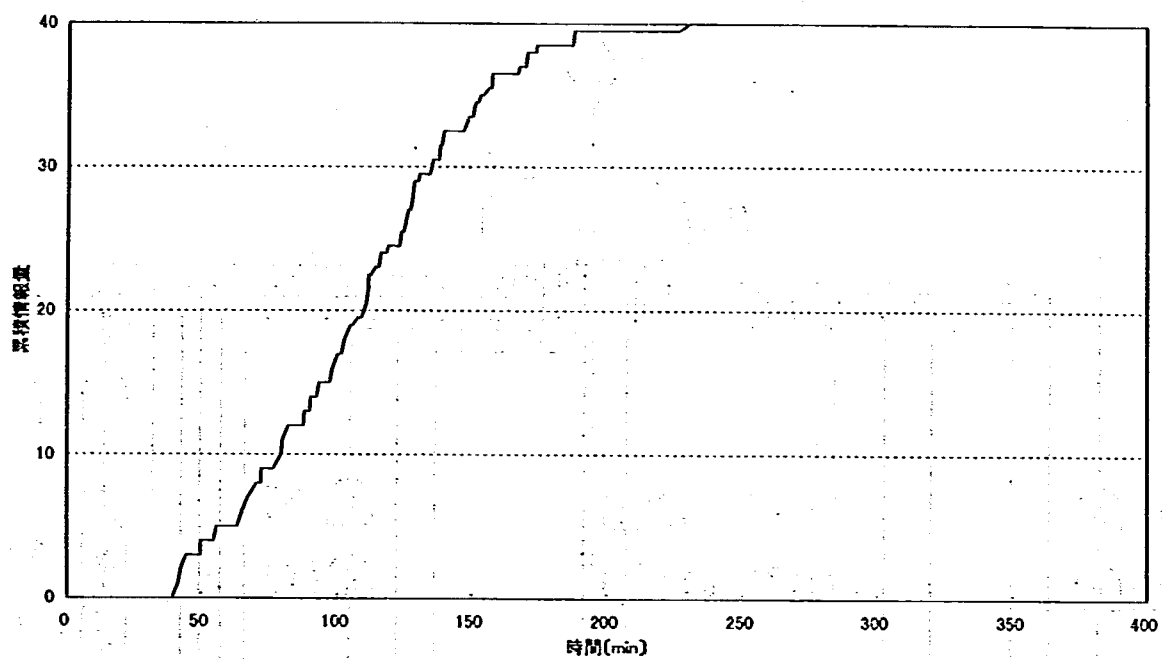
附図 4(b) 情報量蓄積状況(DAY-4)



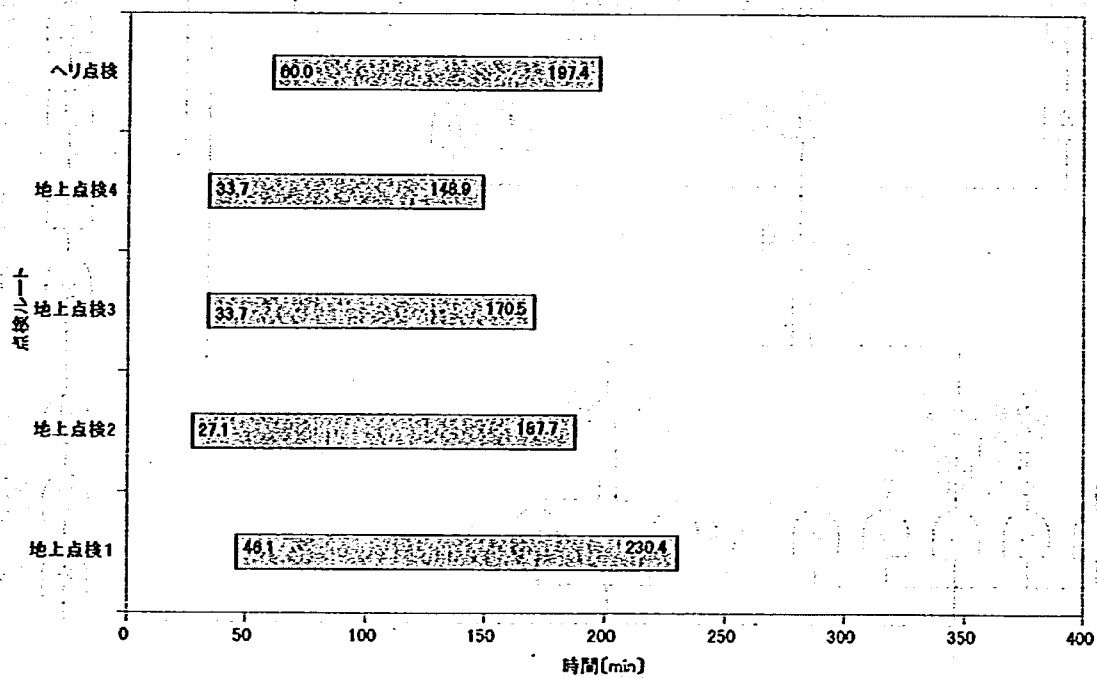
附図 4(c) 点検所用時間(DAY-4)



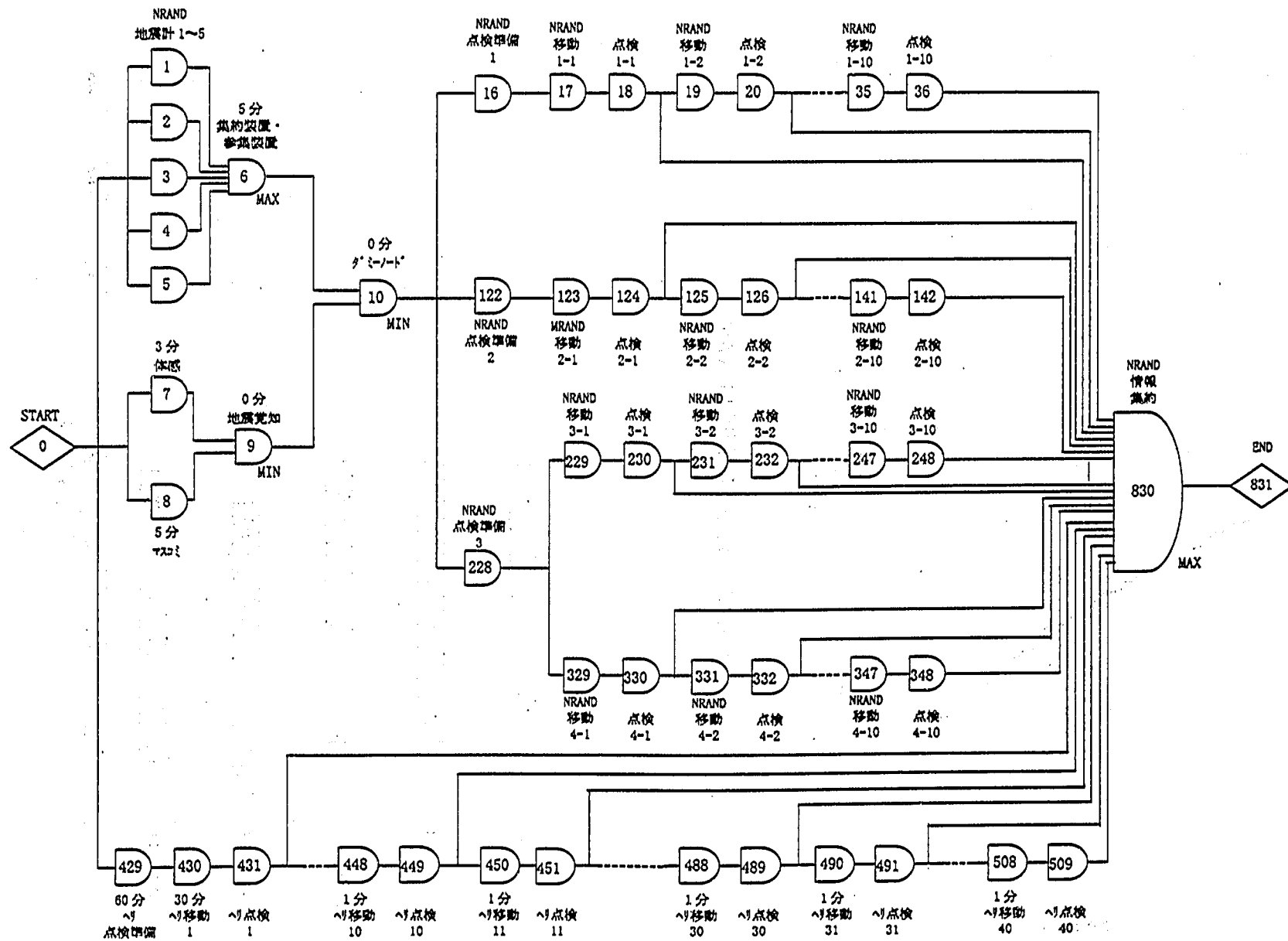
附図5(a) モデルネットワーク (DAY-5)



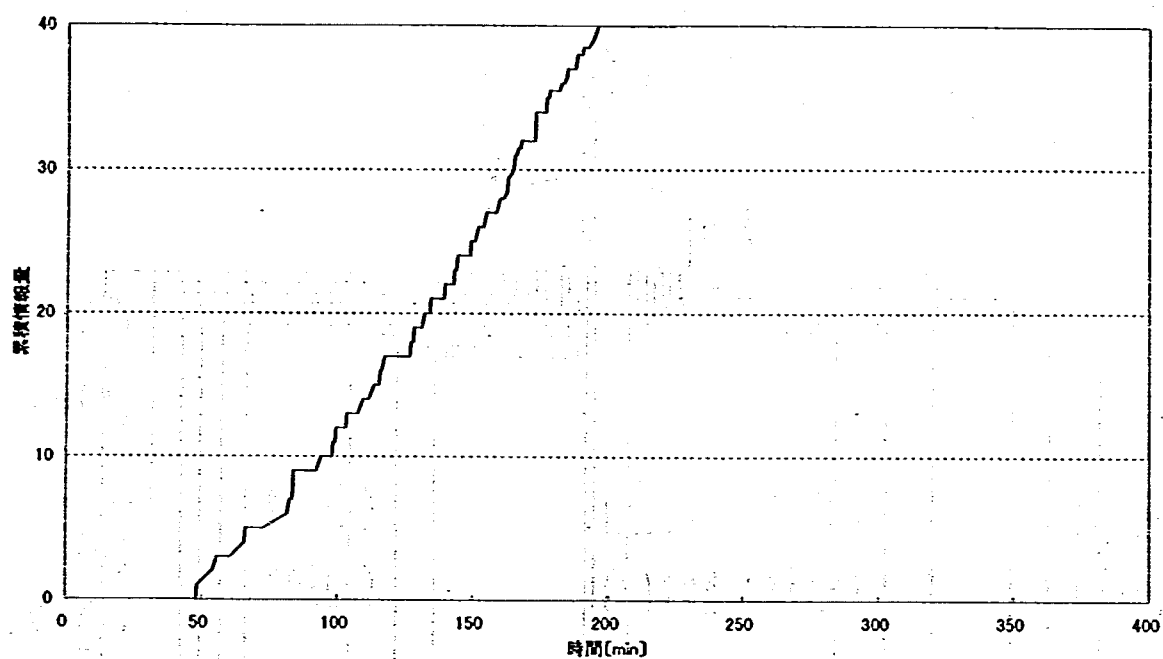
附図 5(b) 情報量蓄積状況(DAY-5)



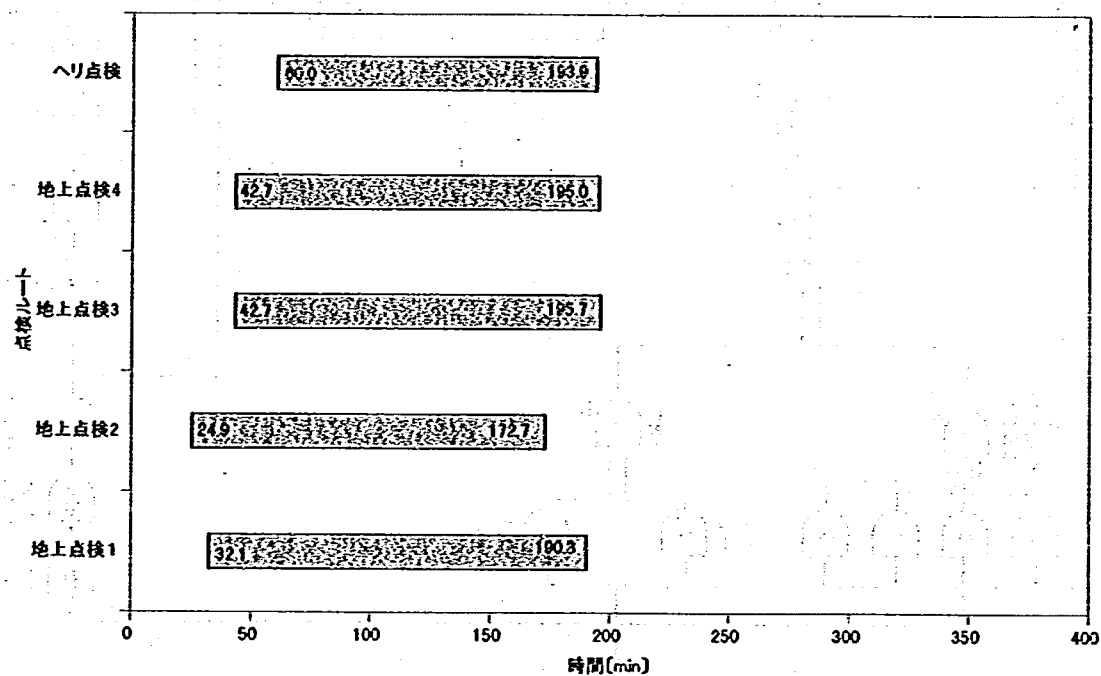
附図 5(c) 点検所用時間(DAY-5)



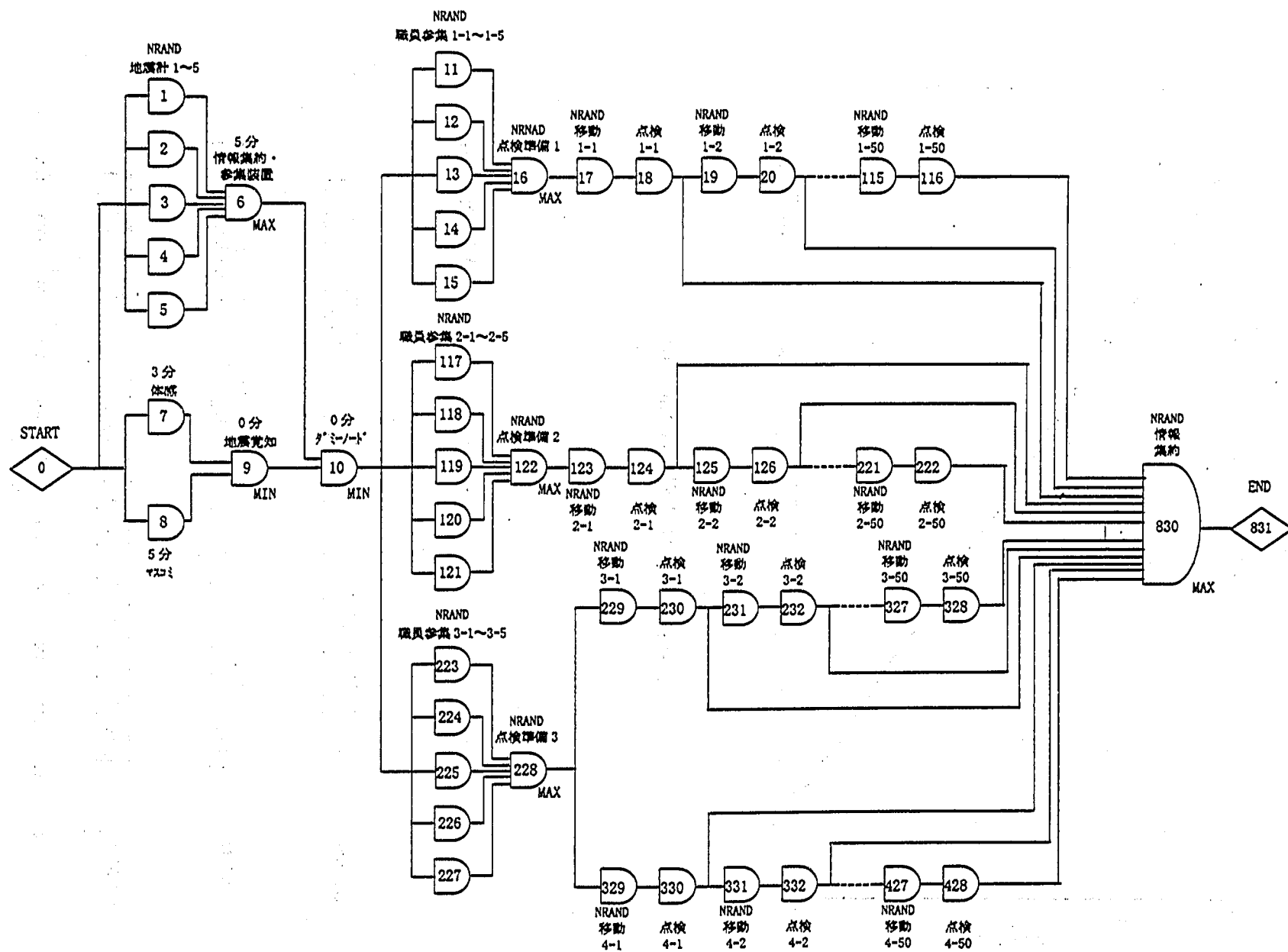
附図 6(a) モデルネットワーク (DAY-6)



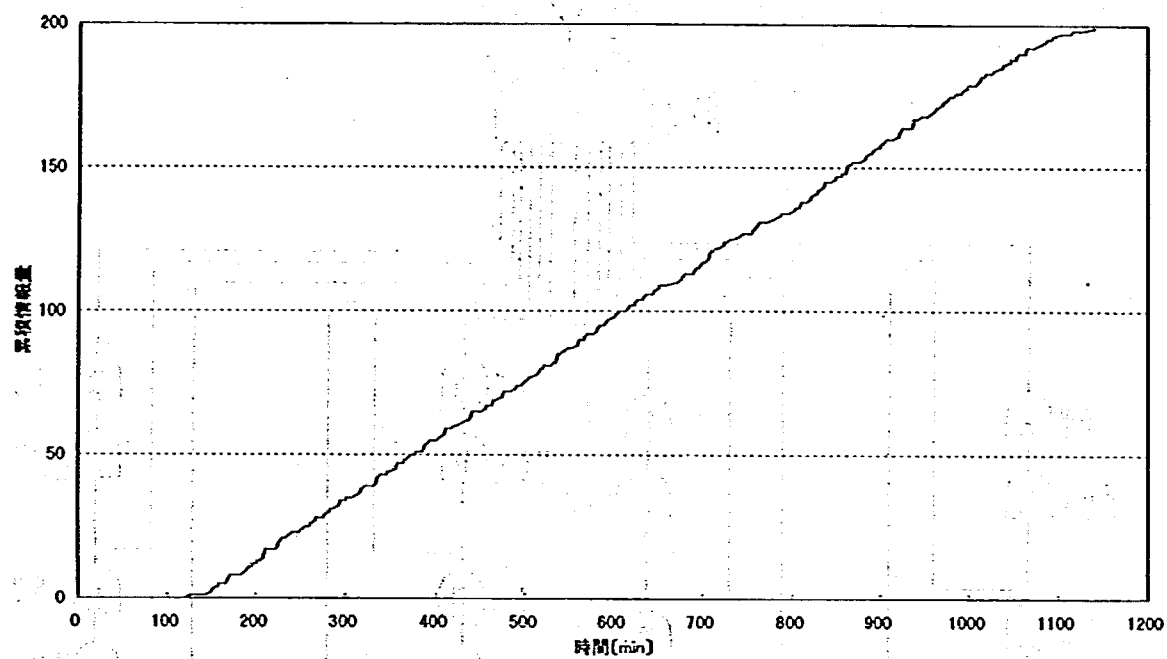
附図 6(b) 情報量蓄積状況(DAY-6)



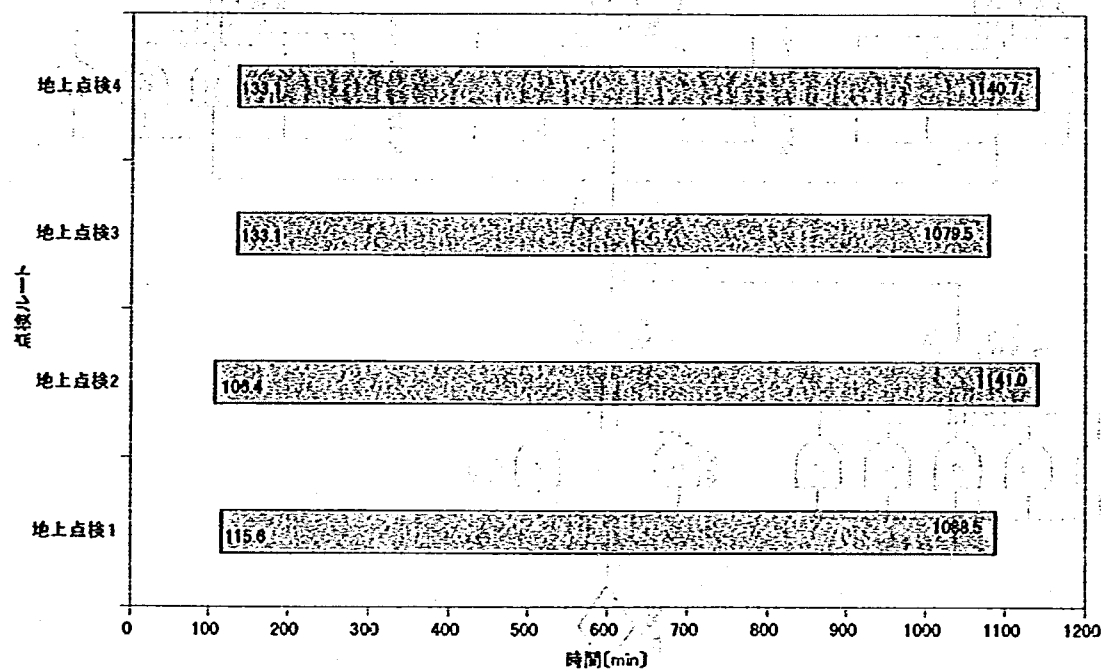
附図 6(c) 点検所用時間(DAY-6)



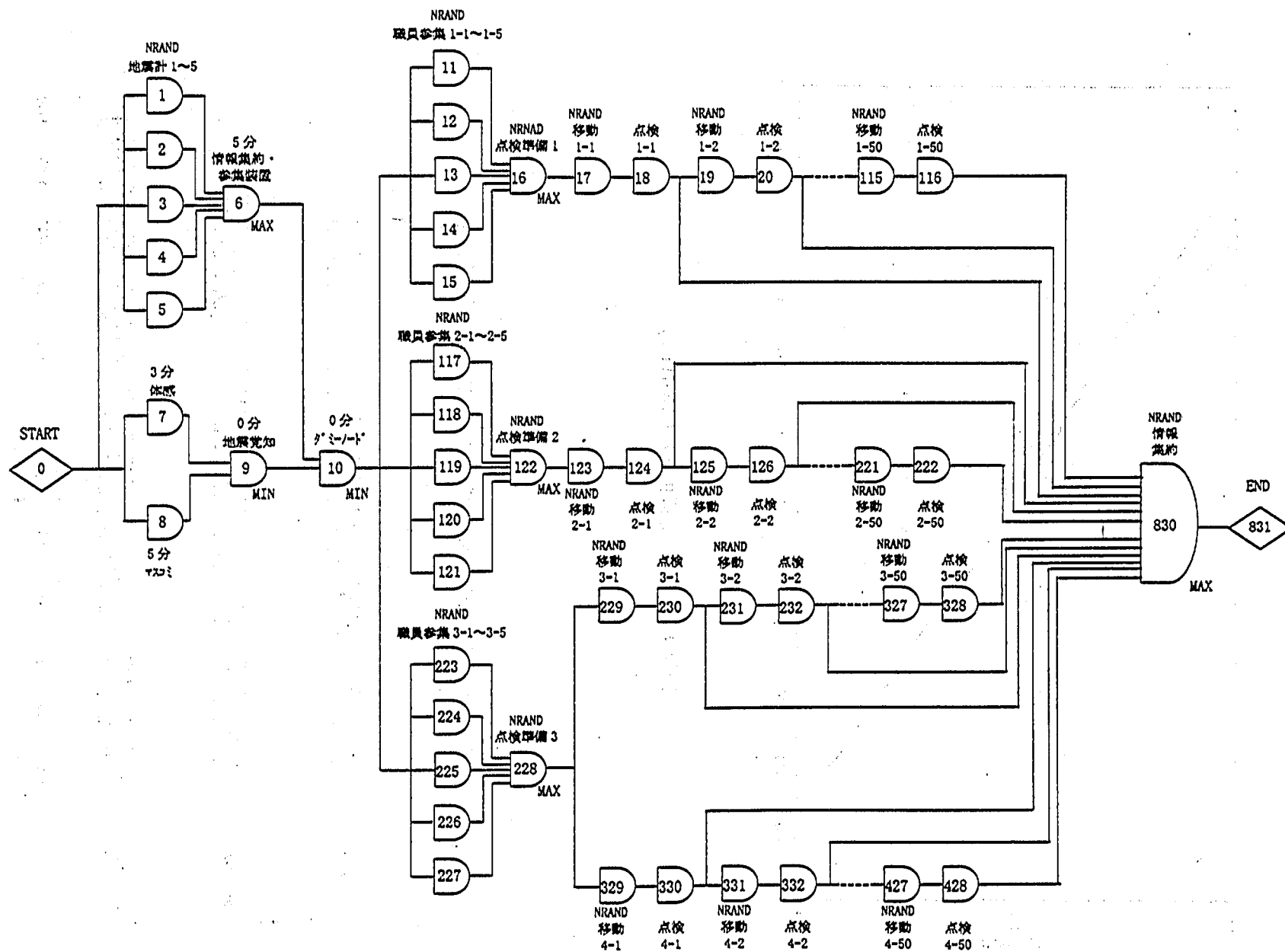
附図 7(a) モデルネットワーク (NIGHT-1)



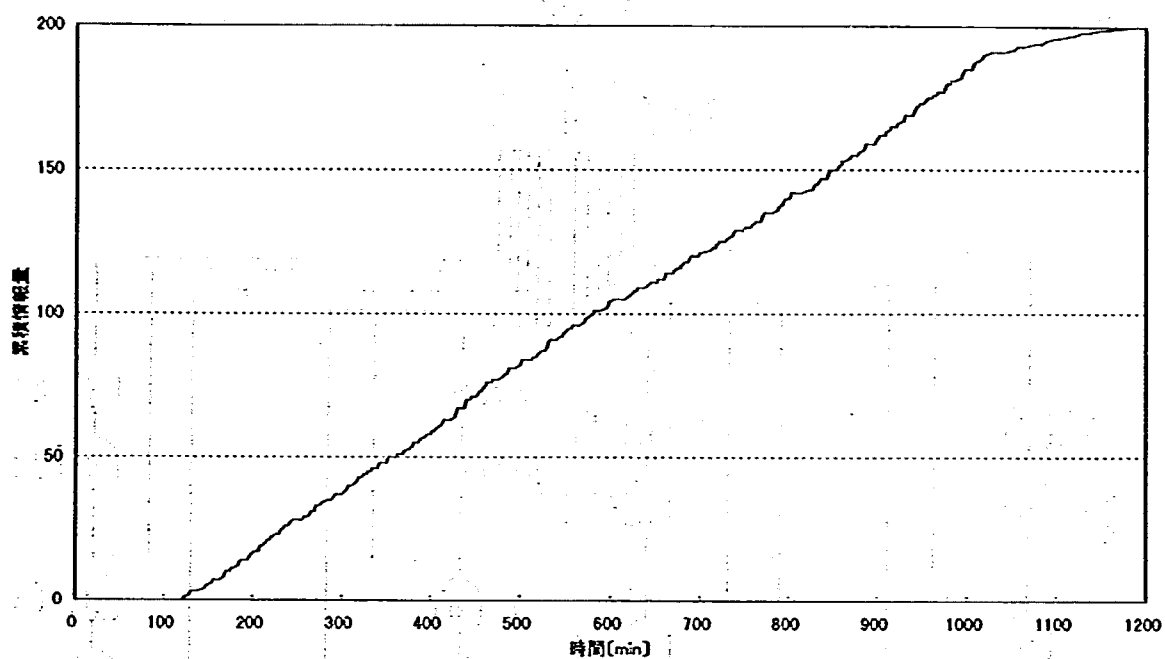
附図 7(b) 情報量蓄積状況(NIGHT-1)



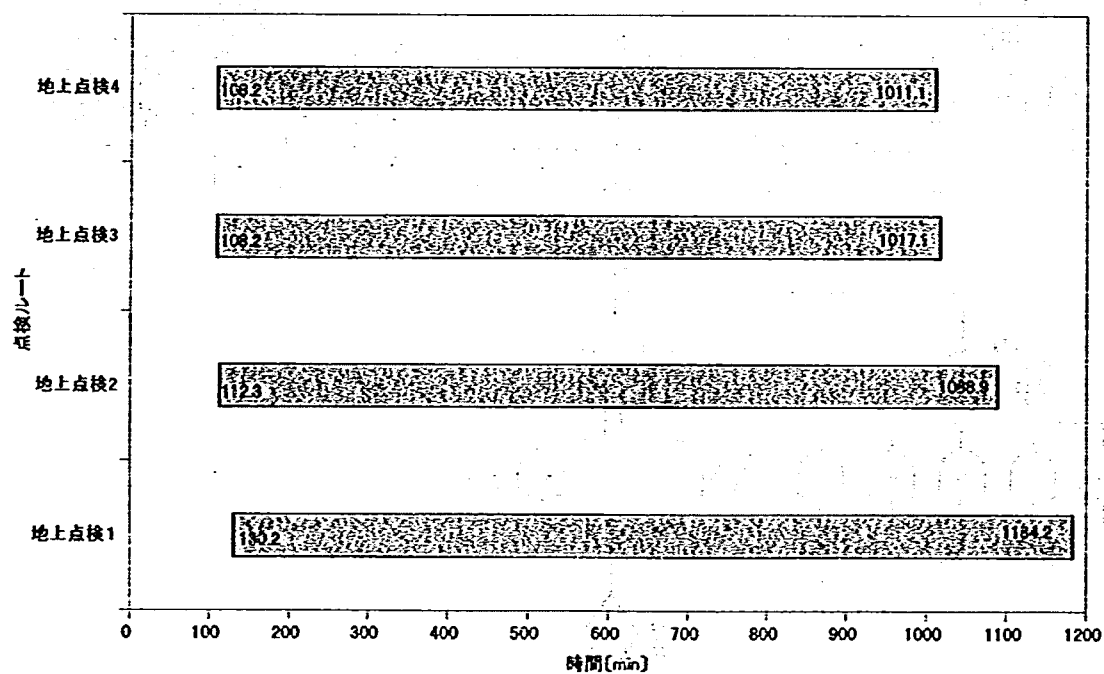
附図 7(c) 点検所用時間(NIGHT-1)



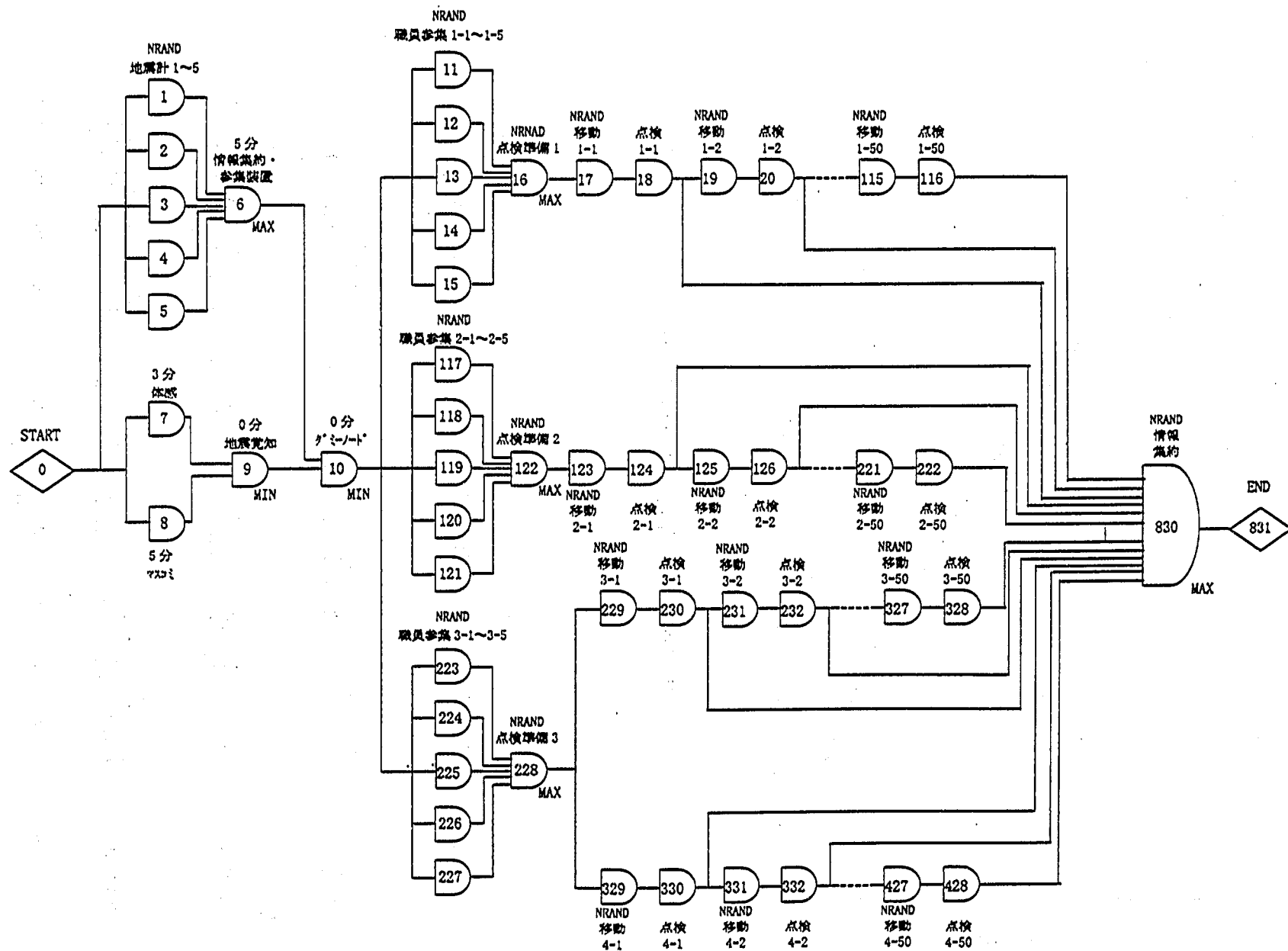
附图 8(a) モデルネットワーク (NIGHT-2)



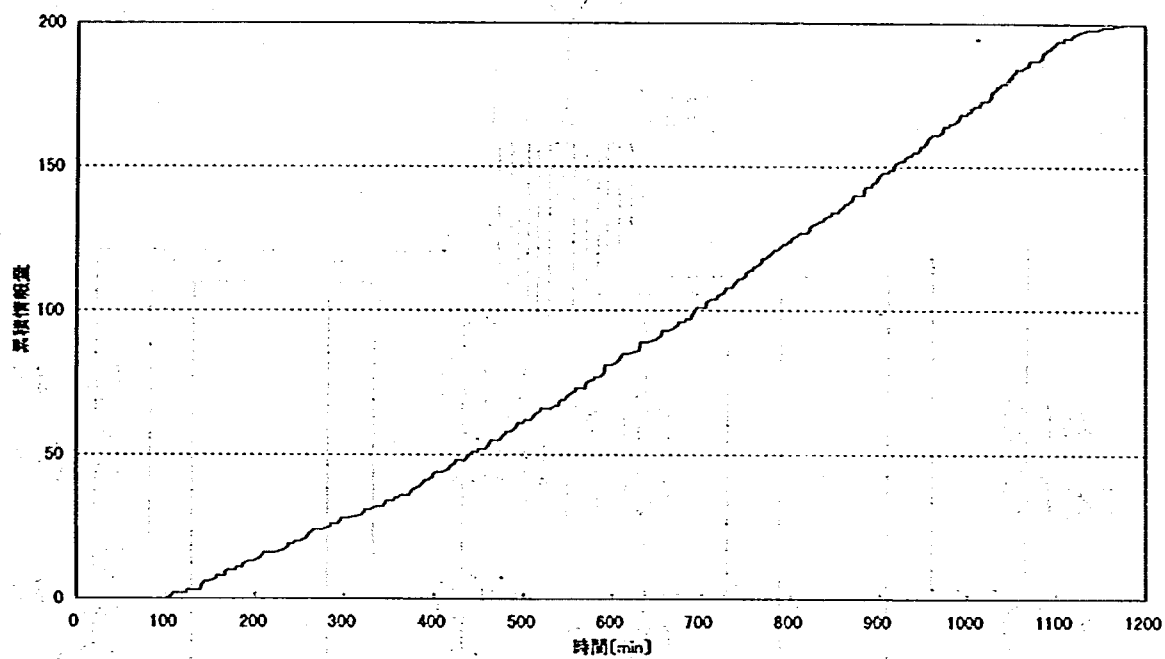
附図 8(b) 情報量蓄積状況(NIGHT-2)



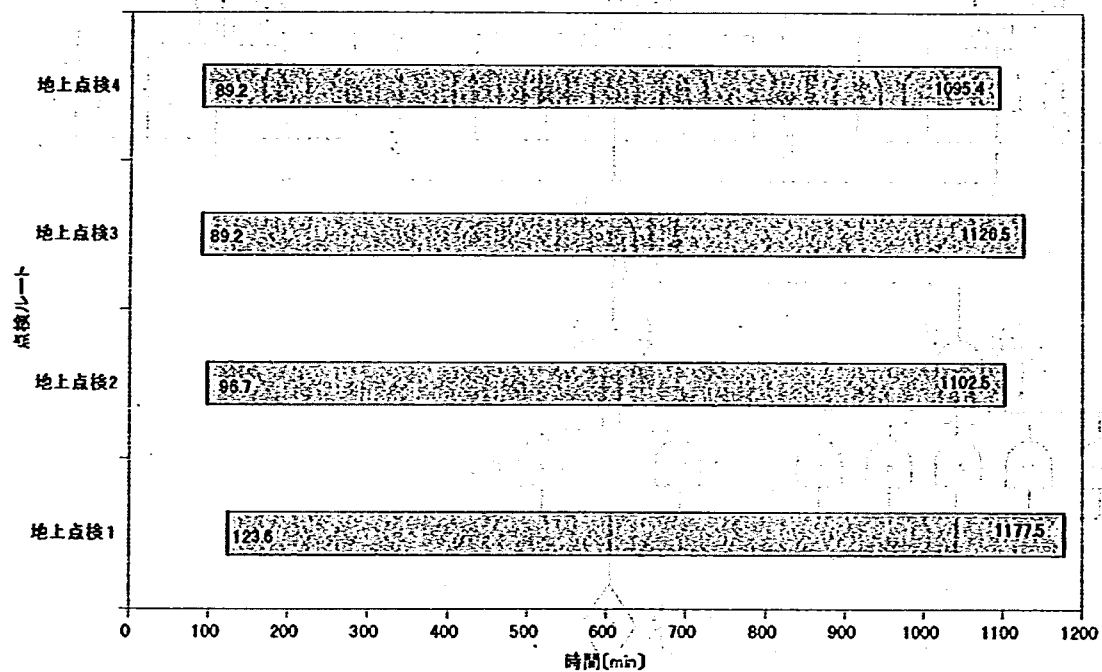
附図 8(c) 点検所用時間(NIGHT-2)



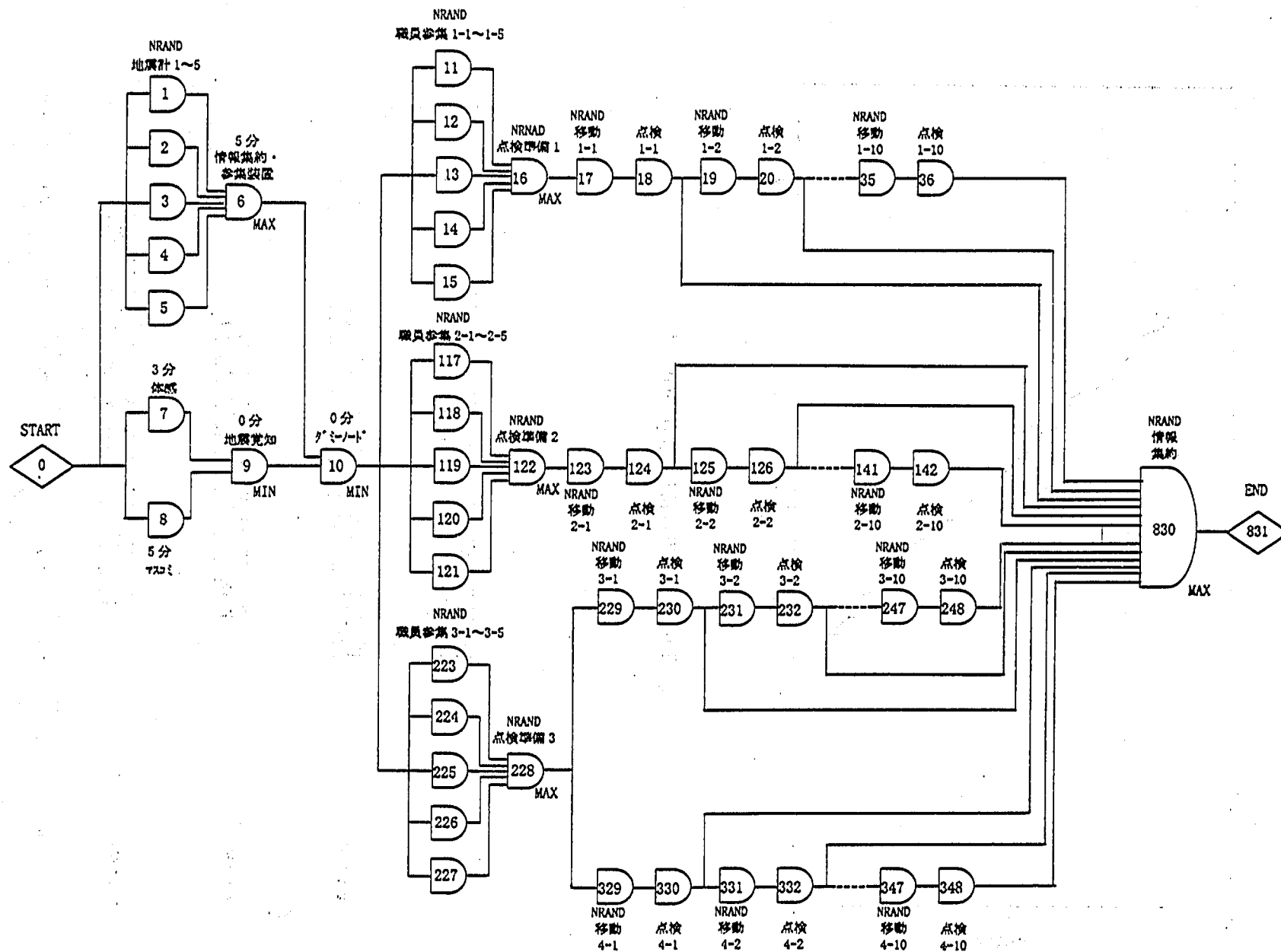
附図 9(a) モデルネットワーク (NIGHT-3)



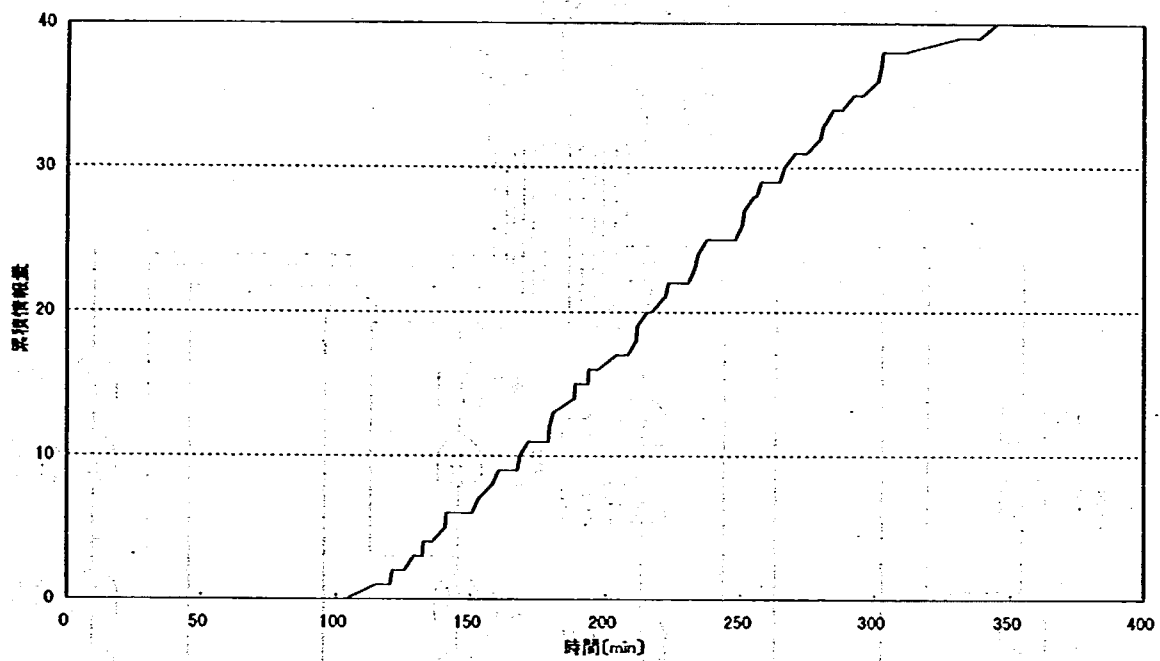
附図 9(b) 情報量蓄積状況(NIGHT-3)



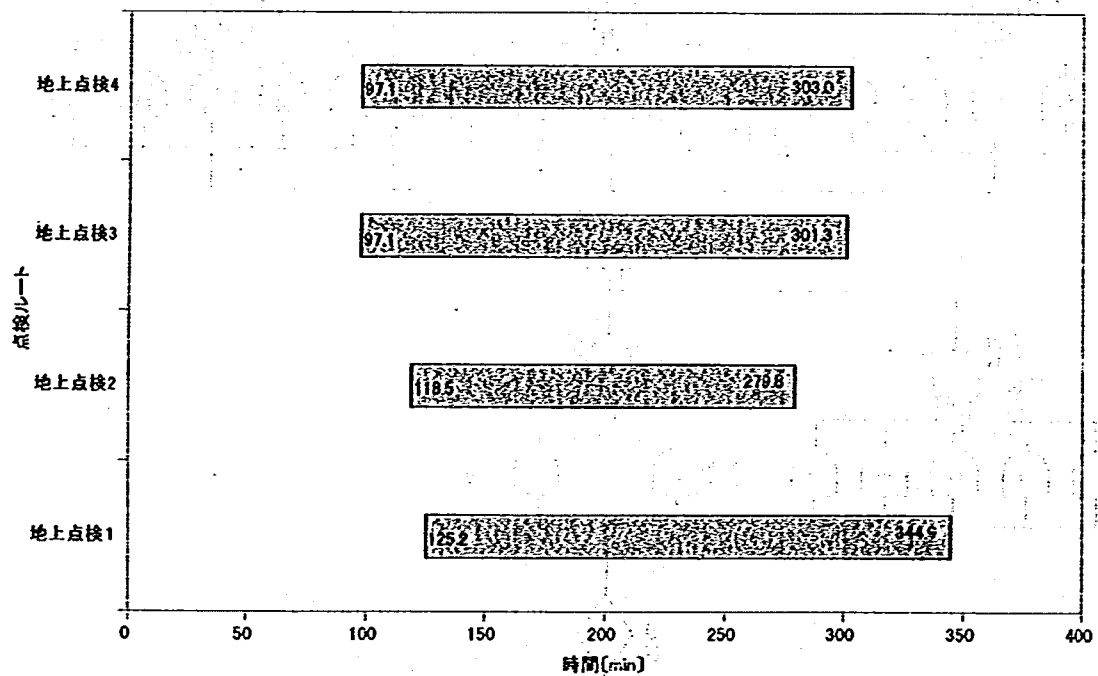
附図 9(c) 点検所用時間(NIGHT-3)



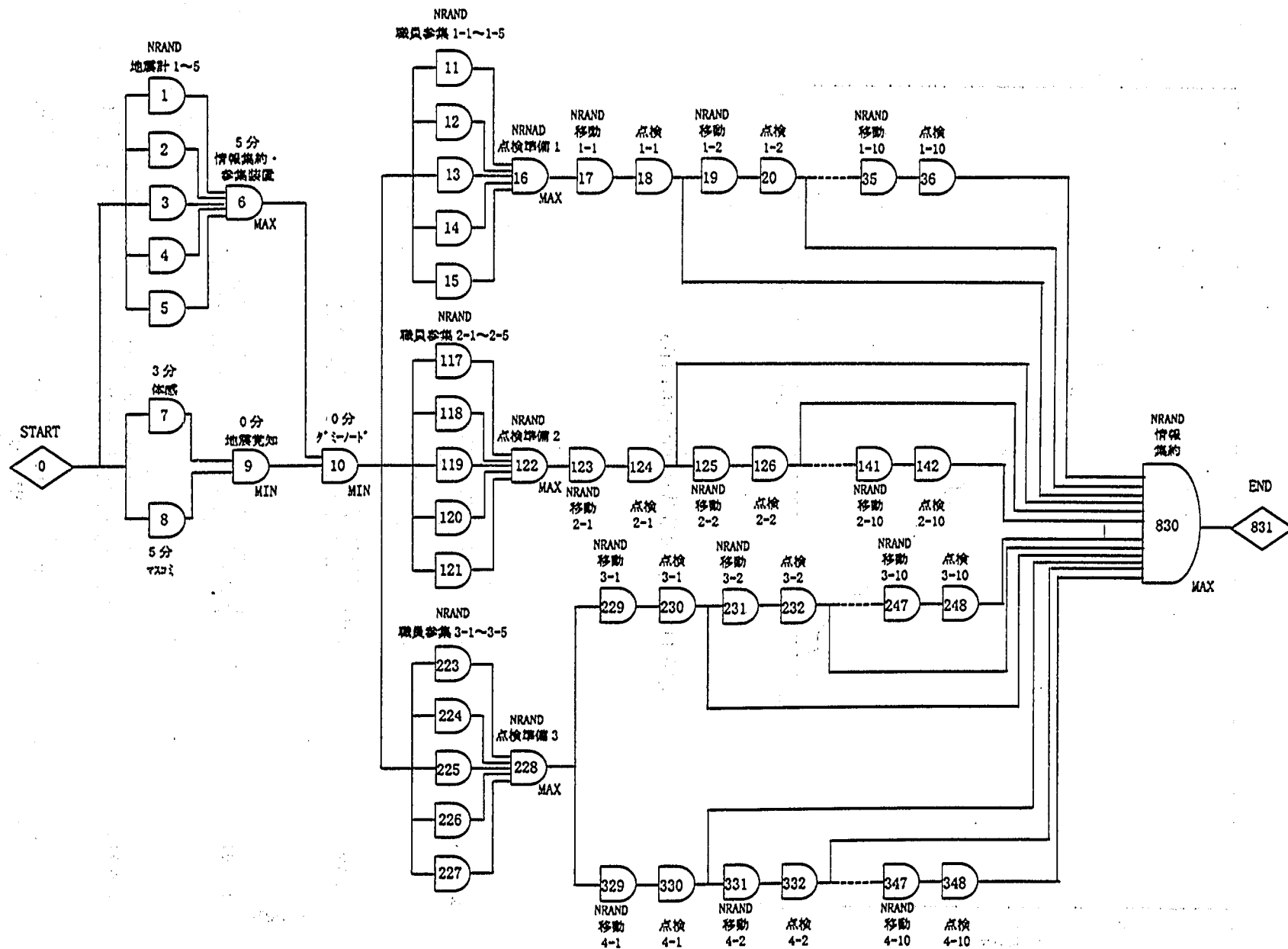
附図 10(a) モデルネットワーク (NIGHT-4)



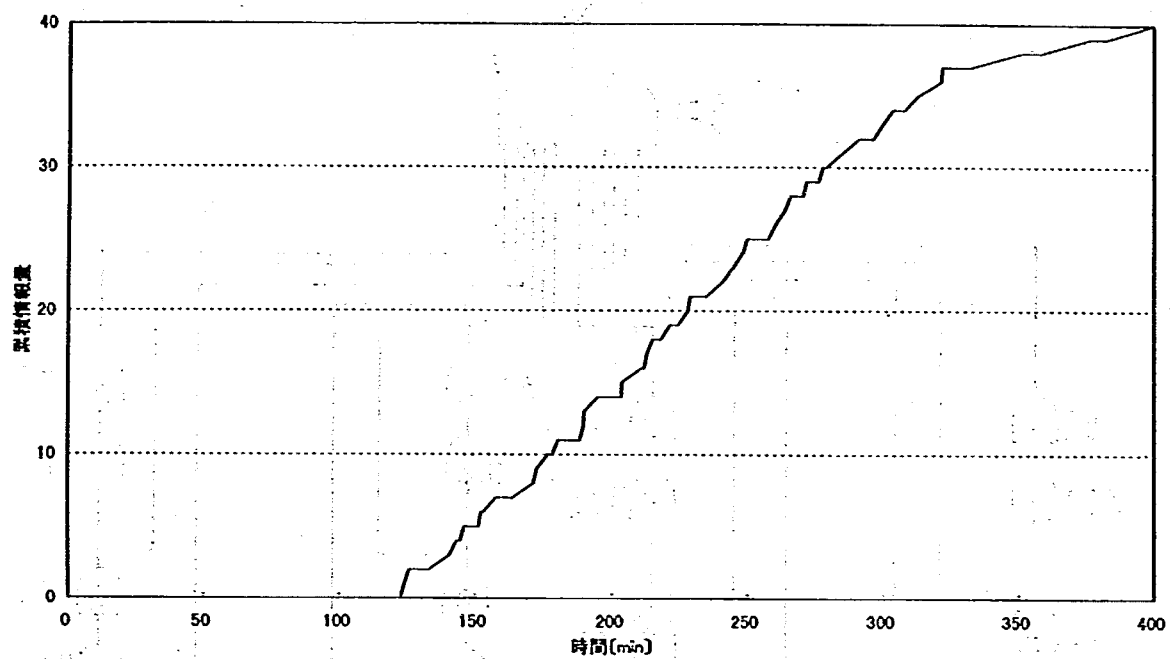
附図 10(b) 情報量蓄積状況(NIGHT-4)



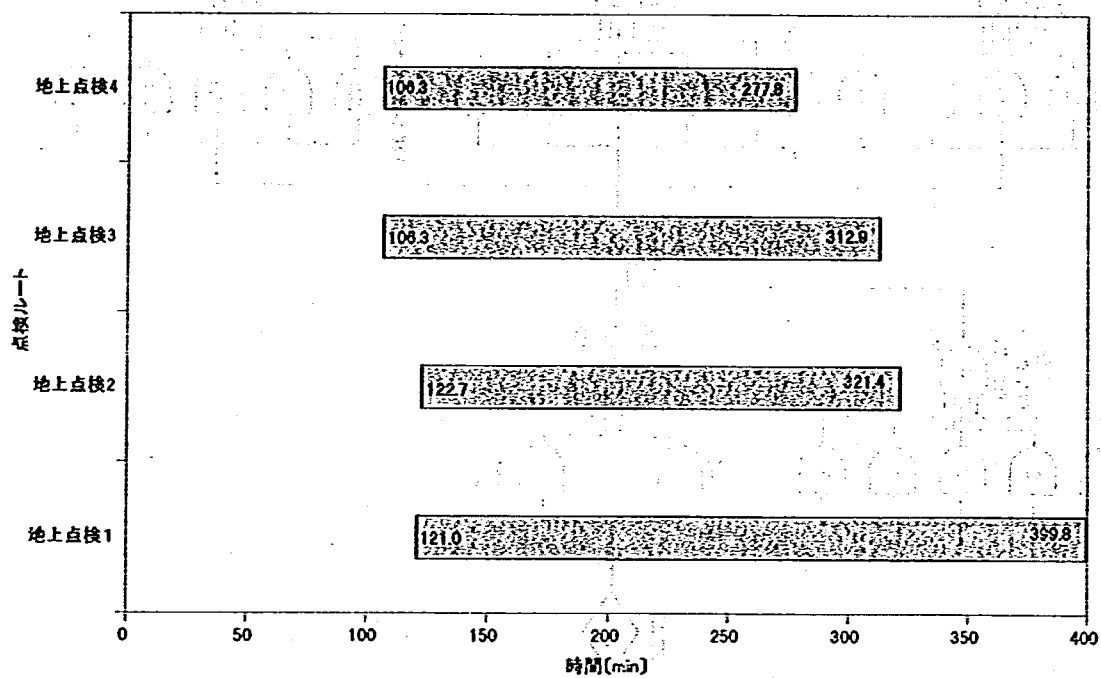
附図 10(c) 点検所用時間(NIGHT-4)



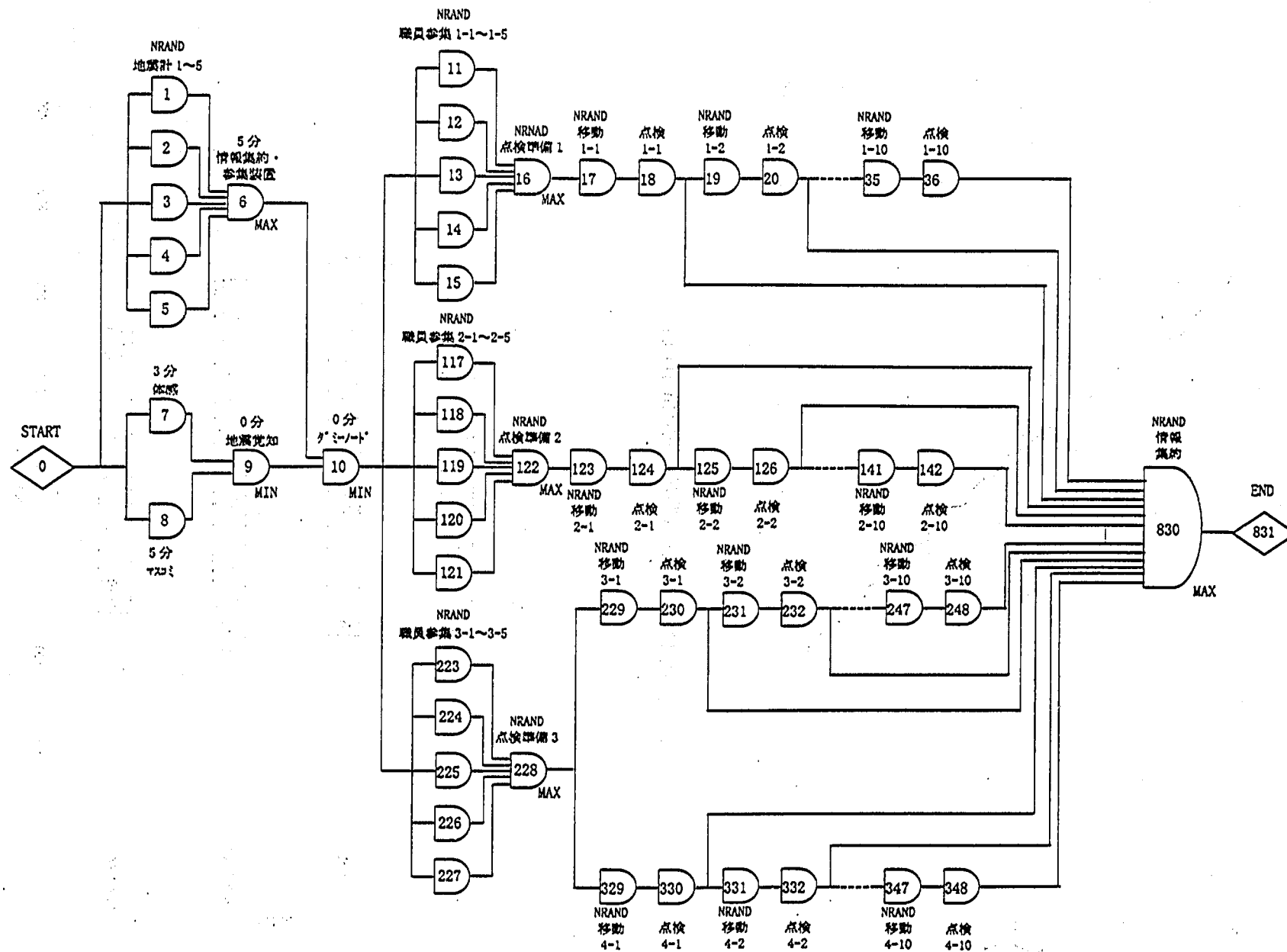
附図 11(a) モデルネットワーク (NIGHT-5)



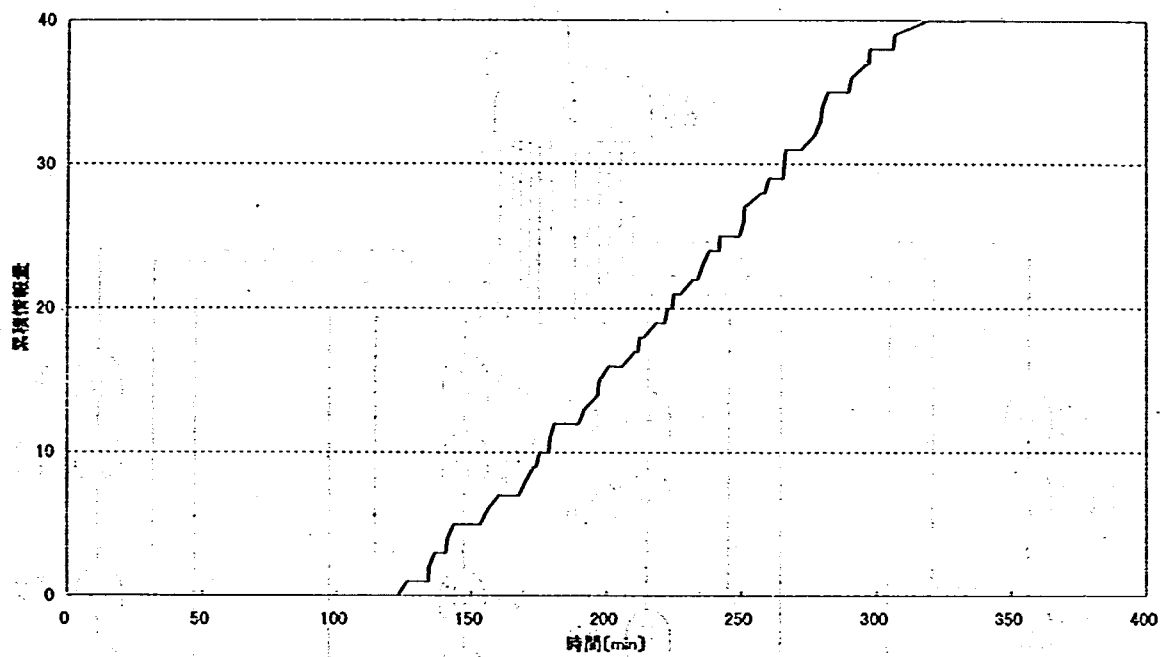
附図 11(b) 情報量蓄積状況(NIGHT-5)



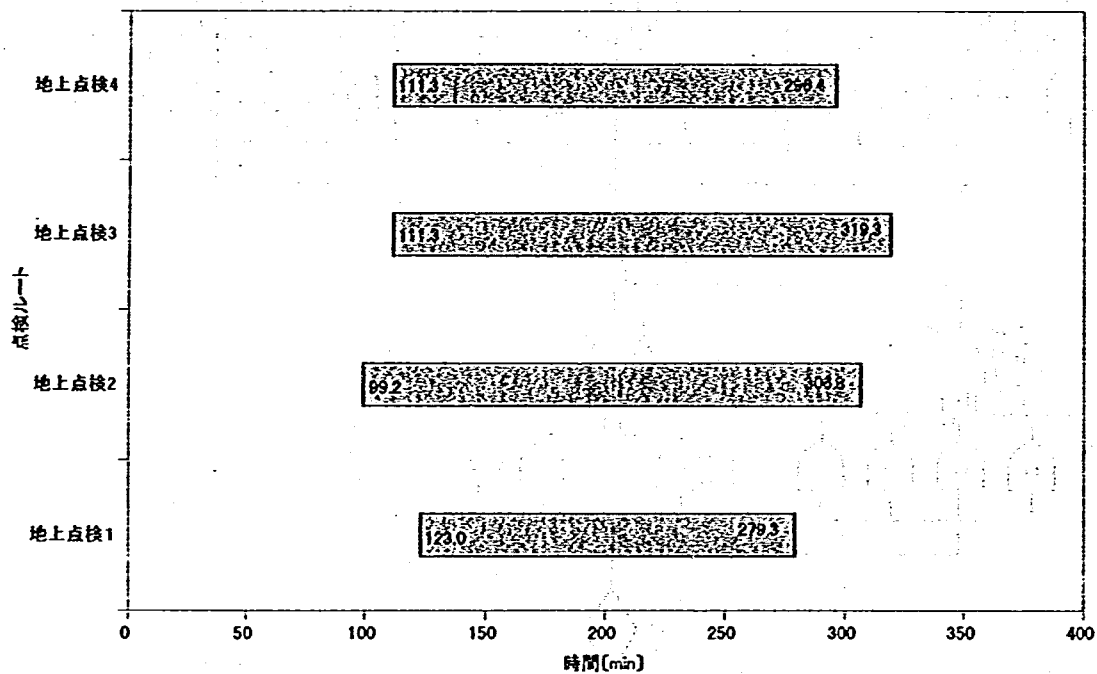
附図 11(c) 点検所用時間(NIGHT-5)



附図 12(a) モデルネットワーク (NIGHT-6)



附図 12(b) 情報量蓄積状況(NIGHT-6)



附図 12(c) 点検所用時間(NIGHT-6)

ソース・コード

ソース・コード：ゲートクラスの宣言・定義

[CGate.h]

```
//////////////////////////////////////////
// Declarations and Definitions for the gate classes.
// This code does not work by itself but must be used with other programs
// that use these classes.
//
// This code is a copyright-reserved program.
// When any portion of the code are changed or used in any document,
// publication and software be sure to print the copyright statement,
// and contact the person below.
// The source code can in the text format is available on your request.
//
// NOZAKI, Tomofumi
// Earthquake Disaster Prevention Technology Division,
// Earthquake Disaster Prevention Research Center,
// Public Works Research Institute, Ministry of Construction
// Phone : +81-298-64-2211(Ext. 4954)
// Fax : +81-298-64-0598
// E-mail: nozaki@pwri.go.jp
//
// July 22, 1998
//////////////////////////////////////////
```

```
//////////////////////////////////////////
//      CInformation
//////////////////////////////////////////
class CInformation{
public:
    double time ;
    double info ;
} ;
```

```
//////////////////////////////////////////
//      CGate
//////////////////////////////////////////
class CGate{
protected:
    CInformation m_Output ;
    int m_Output_Flg ;
    int m_nInput ;
    int m_nInstiution ;
    CGate **m_pInputNode ;
    char m_Caps[64] ;
public:
    CGate(int nInput) ;
    ~CGate() ;
    int GetInputNumber() ;
    CGate* GetInputNodePtr(int nInput) ;
    virtual void SetExecTime(double time, double info) {} ;
```

```

    virtual void SetThresholdTime(double t) {} ;
    void SetInstiution(int no) ;
    int GetInstiution() ;
    virtual CInformation GetOutput()=0 ;
    int GetOutputFlg() ;
    void Connect(CGate* pGate) ;
    char *GetCaps() ;
};

////////////////////////////////////
//      CStartNode
////////////////////////////////////
class CStartNode:public CGate{
public:
    CStartNode(char* name) ;
    CInformation GetOutput() ;
};

////////////////////////////////////
//      CEndNode
////////////////////////////////////
class CEndNode:public CGate{
public:
    CEndNode(char* name) ;
    CInformation GetOutput() ;
};

////////////////////////////////////
//      CWaitMaxGate
////////////////////////////////////
class CWaitMaxGate:public CGate{
    CInformation m_execTime ;
    double m_thresholdTime ;
public:
    CWaitMaxGate(int nInput, char* name) ;
    void SetExecTime(double time, double info) ;
    void SetThresholdTime(double t) ;
    CInformation GetOutput() ;
};

////////////////////////////////////
//      CWaitMinGate
////////////////////////////////////
class CWaitMinGate:public CGate{
    CInformation m_execTime ;
    double m_thresholdTime ;
public:
    CWaitMinGate(int nInput, char* name) ;
    void SetExecTime(double time, double info) ;
    void SetThresholdTime(double t) ;
    CInformation GetOutput() ;
};

```

[CGate.cpp]

```
#include "CGate.h"
```

```
////////////////////////////////////  
//      CGate  
////////////////////////////////////
```

```
CGate::CGate(int nInput)
```

```
{  
    if(nInput){  
        m_pInputNode = new CGate*[nInput] ;  
    }  
    m_nInput = 0 ;  
}
```

```
CGate::~CGate()
```

```
{  
    if(m_nInput){  
        delete m_pInputNode ;  
    }  
}
```

```
int CGate::GetInputNumber()
```

```
{  
    return m_nInput ;  
}
```

```
CGate* CGate::GetInputNodePtr(int i)
```

```
{  
    return m_pInputNode[i] ;  
}
```

```
void CGate::Connect(CGate* pGate)
```

```
{  
    m_pInputNode[m_nInput] = pGate ;  
    m_nInput ++ ;  
}
```

```
void CGate::SetInstiution(int no)
```

```
{  
    m_nInstiution = sino ;  
}
```

```
int CGate::GetInstiution()
```

```
{  
    return m_nInstiution ;  
}
```

```
int CGate::GetOutputFlg()
```

```
{  
    return m_Output_Flg ;  
}
```

```
char* CGate::GetCaps()
```

```
{  
    return m_Caps ;  
}
```

```

////////////////////////////////////
//      CStartNode
////////////////////////////////////
CStartNode::CStartNode(char* name):CGate(0)
{
    strcpy(m_Caps, name);
    m_Output_Flg = OPEN ;
    m_nInstiution = 0 ;
}

CInformation CStartNode::GetOutput()
{
    CInformation dmy ;
    dmy.time = 0.0 ;
    dmy.info = 0.0 ;

    return dmy ;
}

////////////////////////////////////
//      CEndNode
////////////////////////////////////
CEndNode::CEndNode(char* name):CGate(1)
{
    strcpy(m_Caps, name) ;
    m_nInstiution = 0 ;
    m_Output_Flg = CLOSE ;
}

CInformation CEndNode::GetOutput()
{
    if(m_Output_Flg == CLOSE){
        if(GetInputNodePtr(0)->GetOutputFlg() == OPEN){
            m_Output.time = GetInputNodePtr(0)->GetOutput().time ;
            m_Output.info = 0.0 ;
            m_Output_Flg = OPEN ;
        }
    }
    return m_Output ;
}

////////////////////////////////////
//      CWaitMaxGate
////////////////////////////////////
CWaitMaxGate::CWaitMaxGate(int nInput, char* name):CGate(nInput)
{
    strcpy(m_Caps, name);
    m_nInstiution = 0 ;
    m_Output_Flg = CLOSE ;
}

void CWaitMaxGate::SetExecTime(double time, double info)
{
    m_execTime.time = time ;
    m_execTime.info = info ;
}

```

```

void CWaitMaxGate::SetThresholdTime(double t)
{
    m_thresholdTime = t ;
}

```

```

CInformation CWaitMaxGate::GetOutput()
{
    int i ;
    CInformation data, DMax ;
    DMax.time = -999999.9 ;
    DMax.info = 0 ;

    if(m_Output_Flg == OPEN) {
    } else {
        for(i=0; i<m_nInput; i++) {
            if(GetInputNodePtr(i)->GetOutputFlg()==OPEN) {
                data = GetInputNodePtr(i)->GetOutput() ;
            } else {
                return m_Output ;
            }
            if(m_thresholdTime == 0.0
            || data.time <= m_thresholdTime) {
                if(data.time > DMax.time) {
                    DMax.time = data.time ;
                }
                DMax.info += data.info ;
            }
        }
        if(DMax.time < 0.0) {
            m_Output.time = m_thresholdTime ;
            m_Output.info = 0.0 ;
        } else {
            m_Output.time = DMax.time + m_execTime.time ;
            m_Output.info = m_execTime.info ;
        }
        m_Output_Flg = OPEN ;
    }

    return m_Output ;
}

```

```

////////////////////////////////////
//      CWaitMinGate
////////////////////////////////////
CWaitMinGate::CWaitMinGate(int nInput, char* name):CGate(nInput)
{
    strcpy(m_Caps, name);
    m_nInstitution = 0 ;
    m_Output_Flg = CLOSE ;
}

```

```

void CWaitMinGate::SetExecTime(double time, double info)
{
    m_execTime.time = time ;
    m_execTime.info = info ;
}

```

```

void CWaitMinGate::SetThresholdTime(double t)
{
    m_thresholdTime = t ;
}

CInformation CWaitMinGate::GetOutput()
{
    int i ;
    CInformation data, DMin ;
    DMin.time = 9999999.9 ;

    if(m_Output_Flg == OPEN){
    } else {
        for(i=0; i<m_nInput; i++){
            if(GetInputNodePtr(i)->GetOutputFlg()==OPEN){
                data = GetInputNodePtr(i)->GetOutput() ;
            } else {
                return m_Output ;
            }
            if(data.time < DMin.time) DMin = data ;
        }
        if(m_thresholdTime == 0.0
        || DMin.time <= m_thresholdTime){
            m_Output.time = DMin.time + m_execTime.time ;
            m_Output.info = m_execTime.info ;
        } else {
            m_Output.time = m_thresholdTime ;
            m_Output.info = 0.0 ;
        }
        m_Output_Flg = OPEN ;
    }

    return m_Output ;
}

```