数理解析研究所講究録
第596巻 1986年 179-193

流体近似による待ち行列の解析とパソコン用ソフトＦＦＱＡ

東京工業大学　森村英典 (Hidenori Morimura)
同　Ｉ．Ｍ．プレマチャンドラ (I.M. Premachandra)

１．流体近似による待ち行列解析の実用性

窓口が暇になることがあまり起こらないまま過程が進行するようなタイプの待ち行列システムの実例は多い。たとえば、鉱石運搬船は満載してきた鉱石を艀を使って順次荷下しするが、すべての鉱石を下し終えれば、その過程は終了する。その間、艀やタグボートの不足のため、荷下しを中断することはあるが、そう頻繁に起こるわけではない。生産工程では、生産を休まずに続行するため、仕掛り在庫をある程度持つようにしているが、それがあまり大きくなると不経済であるから、仕掛り在庫の適正量を知りたいという要望が多い。

このような問題に対しては、確率論的なアプローチを中心とする通常の待ち行列理論によるよりも、いわゆる流体近似により、確定的な問題として扱う方がよいことが知られている。この方法は確率論的方法に比べ原理的にははるかに簡単であるとはいえ、種々のモデルに対して、個々に計算を実行しようとするとそれなりの工夫も必要で、それがネックになって現場で使いこなされているとは言えない状況である。

そこで、現実に多いタイプの待ち行列システムを定式化できる包括的なモデルを考え、それに対する計算のアルゴリズムを確立し、更にそれを利用して現実問題をこのモデルで定式化するための使いやすい支援システムをパソ

コン上で作ることは実用上有意義な仕事であると言えよう。

この小論のねらいもそこにあり、支援システムのプロトタイプとなりうるプログラム・パッケージ FFQA を作成したという報告であるが、ここには、その基本的な考え方を中心に述べる。よりテクニカルな内容については〔3〕を、また FFQA の具体的な扱い方は〔2〕のマニュアルを参照して頂きたい。

2．基本となる考え方と記号

1．で述べたタイプの問題を取り扱うためには、サービス・ステーションは何段にも配列されているものを取り扱わなければならない。各段のサービス・ステーションにおける待ち行列は、Newell流の

$$Q(t) = A(t) - D(t)$$

という形で表現する。ここで、A(t)は累積到着数、D(t)は累積退去数を表す。

各段のサービス・ステーションにおける待ち行列であることを示すためにこれらの記号にはサービス・ステーションの番号を示す添字を付ける。タンデム型の待ち行列では、たとえば第1段の退去過程は第2段の到着過程になるから

$$D_1(t) = A_2(t) \qquad (1)$$

が成り立つ。

単一のものが次々のサービス・ステーションを流れて行くのであれば、このような形の等式で繋いで行くことにより、全体の待ち行列システムを表現することが可能である。しかし上述の鉱石運搬船の例のように、あるステーションにおける客は鉱石で server は艀であるが、次のステーションにおける客は艀で server はタグボートであるというような場合には、艀は艀としての流れを追わなければならないから、（1）のような形の等式が常に成り

立つとは限らない。そこで、通常のサービス・ステーションにおける待ち行列を考えるのではなくて、客の種類ごとのそれを考えることにする。このため、それらをノードと呼んで一般のサービス・ステーションと区別する。また、ステーションごとに変る客を区別するために、客を commodityと呼んでおく。

図１はノードを用いて上述の鉱石運搬船の例をモデル化したものである。このように、いくつかの commodity ごとの流れとして表現するとシステムが容易に書き表される。つまり、第１の commodity として鉱石を考え、第２、第３の commodity として艀とタグボートを考えている。

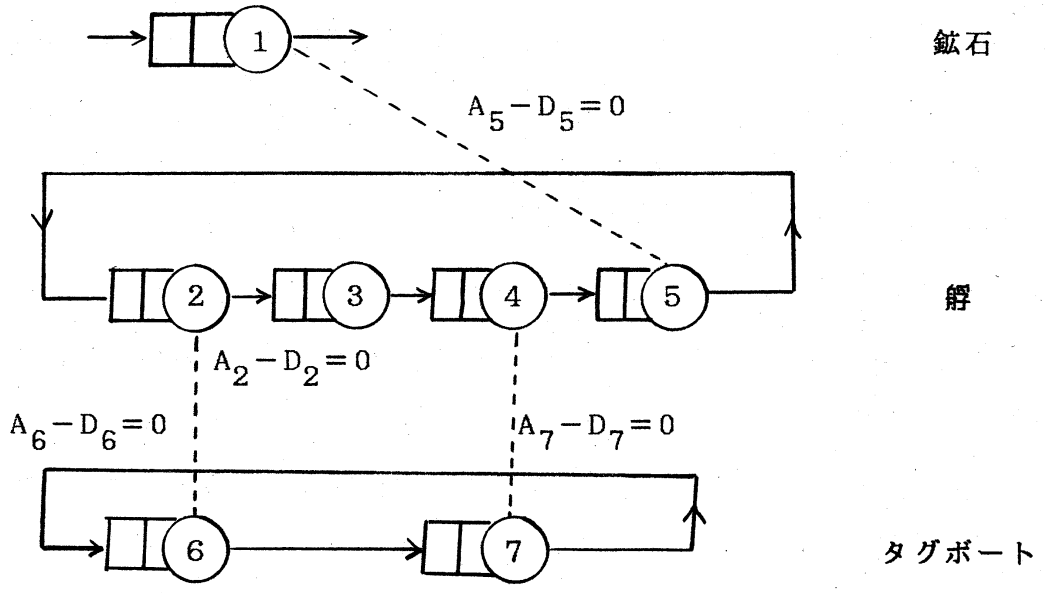

図 1　ノードの設定　（鉱石運搬船の例）

鉱石はノード１だけでサービスを受けるが、艀は２から５までのノードで満載されたり空になったりしながらサービスを受けて行く。そして、タグボートはノード６と７とで客としての役割を持つ。こういう状況を自然な形で表現するのがノード導入の狙いである。

このような図で表現すると、当然ノード間には関係が生ずる。それらを記述するのが制約条件である。たとえばノード１のサービスが行われるためには空の艀が必要であるから、ノード５における客数が正でなければならない。言い換えると、もし

$$A_5 - D_5 = 0 \qquad (2)$$

が成立するときはノード１のサービスは中断されなければならない。（２）のような条件を制約条件と呼ぶ。そして（２）が成立するとき、ノード１のサービスの中断を示すため、indicator variable の値を 1 にする。

indicator variable とは

$CC(i,j) = \begin{cases} 1, & \text{ノード } i \text{ の } j \text{ 番目の制約条件が成立するとき} \\ 0, & \text{その他} \end{cases}$

と定義される変数である。各ノード i では $\Sigma\ CC(i,j)$ の値を常に観察していて、それが 1 になったらノード i からの出力を一時停止して、この値が 0 になるのを待つというのが基本的な考えである。その際、過程の進行を促すために後述の induced delay をこのアルゴリズムの中で自動的に設定する必要がある。

なお、たとえば艀やタグボートが移動するためには時間を要するからそれを記述するために additional node、batch service を表現するために accumulation node、流れの始りを指示するために initial node、といった概念を導入する。これらについては次節で若干の解説をする。

3．一般的アルゴリズムとプログラム・パッケージ FFQA-GAP

このプログラム・パッケージの開発に当たって、まずはできるだけ広い適用対象を確保しておきたいという願望がある。そこで、一般的計算アルゴリズムはその適用対象にできるだけ自由度を持たせることを狙った。そのため、制約条件としては原則としてどのようなものでもよいとしている。勿論、各ノードにおける客数や待ちについての情報は、A（t）やD（t）に盛りこまれたものに限られるから、それ以上の情報を必要とする制約条件は取り扱えないのは当然である。

制約条件としてどのようなものも許すために、プログラム・パッケージにおける制約条件の記述はユーザが BASIC で記述するという形式を取っている。そして、ここに自由度を持たせたために、このアルゴリズムは微小時間ずつ時間を進めながら計算を行っていくという、いわゆる時間駆動方式を採用している。

このようなアルゴリズムを利用して作成したプログラム・パッケージを FFQA-GAP と呼ぶ。FFQA とは Fluid Flow Queueing Analyser の略のつもりで、GAP とは General APproach の略のつもりである。 FFQA には次節で述べる FFQA-RAPID と呼ぶプログラム・パッケージもある。

FFQAを起動すると、入力データをソフト側で聞いてくる。まず、DATA 1 として基礎データを入れる。内容は次の８つである。

1） 計算を進めるための時間増加量 δ （default の場合は 0.25）

2） ノード番号

3） サービス速度

4） t＝0 のときの系内数

5） input set （そのノードに流入する流れを出すノードの集合）

6） output set （そのノードから流出する流れが入るノードの集合）

7) output set が複数個の元を持つとき、各ノードに入る流れの割合

8) 流出する流れが次のノードに入るまでの遅れ

これらのデータは、2)から8)まで、ノードごとに繰り返し聞かれる。

最後の 8)のデータが入力されると、計算機の側では additional node を自動的に設定する。このノードへの入力過程は前のノードの出力過程と同じであるが、出力過程は 8) で入力されたデータ分だけ平行にシフトしたものとなる。また、additional node の番号は、上で入力されたノードの最後の番号に付け加えて、計算機の側で自動的に付けておく。この情報は、summary report の中で印刷出力される。

つぎに、計算機の側では上の 5) の情報からinitial node が何かを知るが、それに対する入力過程を知りたいので、その入力情報を求めてくる。

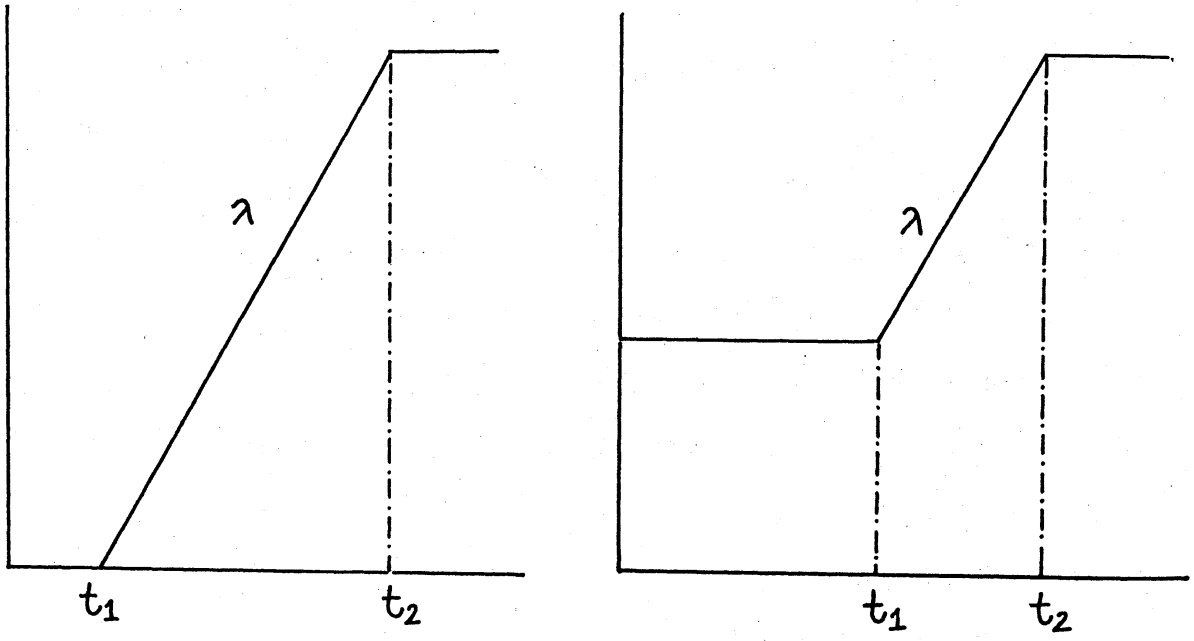

図 2　入力過程のパターン

任意のノード i における到着過程$A_i(t)$は、任意の曲線で表現できればそれに越したことはないが、計算アルゴリズム上いささか厄介である。そこで、当面は図2に示すような形のものだけを想定し、それが入力しやすいよう会話的ソフトにしてある。すなわち、initial node ごとにスクリーンの上にいくつかの入力過程のパターンを示し、ユーザがそれを定めると、それを表現するためのパラメータをスクリーン上の入力過程のパターンの傍に入力するよう求めてくるから、ユーザは簡単に入力できる。

入力過程のパターンにもっと自由度を持たせることはおそらく必要であろう。少なくとも、区分的線形な関数の形は取り扱えるようにしておきたいと考えている。

最後に DATA 3 によって制約条件の入力が求められる。FFQA-GAP では、制約条件の記述はユーザが BASIC を用いて行わなければならないが、それが幾分なりとも容易になるように、いくつかの変数は予め定義してある。ユーザがそれらを利用すると、制約条件の記述は比較的簡単になる。

また、過程の終了を示すための条件、たとえば鉱石運搬船の例ではすべての鉱石を下し終えればその過程は終了すること、を計算機に教えておかなければならない。これも、DATA 3 の中でその入力が求められている。

計算アルゴリズムの基本は上述のとおりで、いわゆる時間駆動方式により、微小時間δずつ時間を進めながら、そして initial node から始めて順次下流のノードへと下りながら、各ノードにおけるA(t)やD(t)およびその間に挟まれた部分の面積などの計算を行っていく。

DATA 3 の中で入力した過程の終了条件により過程が終了すると、FFQA は計算結果を出力する。ここで印刷される summary report は

1) すべての入力情報

2) 各ノードごとの平均滞在時間

3) 各ノードごとの累積到着数と累積退去数

4) 各ノードごとの客数の最大値、最小値および平均値

である。

4．高速化アルゴリズムとプログラム・パッケージ FFQA-RAPID

上のアルゴリズムは微小時間δずつ時間を進めながら計算を行っていく時間駆動方式を採用しているため、計算時間が長くなりがちな欠点がある。そこで、適用対象を絞ることにより、事象駆動方式に改めたのがこのアルゴリズムであり、且つそれを基礎とするプログラム・パッケージ FFQA-RAPID である。この変更に伴い、BASIC 記述による入力も不必要になっている。その結果、適用対象は若干限られるものの、パソコンへの入力は完全に会話的且つ自然に進められ、シミュレーション言語やパソコンに馴染みの薄いユーザにも使いやすくなっている。

なお、現在作られている FFQA-RAPID ではネットワークが扱えないが、そこまで拡張することは比較的容易であるので、近い将来それを試みるつもりである。

基本となるアルゴリズムを以下に述べておく。つぎの記号を用いる。

R_i ：ノード i についての制約条件の個数

t_{ij}：ノード i についての j 番目の制約条件

[STEP 1] $i = 1, 2, \cdots, M$ の順に i を動かして

$$t_i^* = \mathrm{Min}_j\ t_{ij}$$

を計算する。

[STEP 2]

$$t^* = \mathrm{Min}_i\ t_i^*$$

を計算し、各ノード i について t^*までの A_iと D_i との間の面積増加分を求め、いままでの値に加える。（初期値は 0 ）

[STEP 3]　　各ノード i について $\Sigma_j CC(i,j)$ の値を求め、$\Sigma_j CC(i,j) \neq 0$ であるようなノード i の集合 S を作る。Sに属するすべての i に対して

$$t_i^* = t^*$$

と置き、induced delay を導入の後

$$t^* = 0$$

と置きなおす。 STEP 1 へ戻る。

[注] induced delay とは、系内数が 0 になって過程が一時中断したような場合、その時点から再び過程を動かそうとしても直ちに系内数が 0 になって過程は再び中断してしまい計算を続行することが出来なくなることを避けるためと、近似そのものの精度向上のために導入するもので、実際の待ち行列過程では、つぎの到着までの空時間とその到着客のサービス時間の和に相当する時間内には退去過程は始らないという事実を反映するため、

$$(1/\mu_{ai} + 1/\mu_{di}) / 2$$

だけ遅らせることとしている。ここで、$1/\mu_{ai}$ 、$1/\mu_{di}$ はそれぞれ、平均到着間隔および平均サービス時間を表す。

また、ブロッキングの例のように、系内数がある値になって過程が一時中断したような場合における induced delay は、ブロッキングの原因となったノードの平均サービス時間 $1/\mu_{dj}$ として設定してある。

以上のアルゴリズムの具体的な演算方法を例示するため、図 3 の場合について説明する。

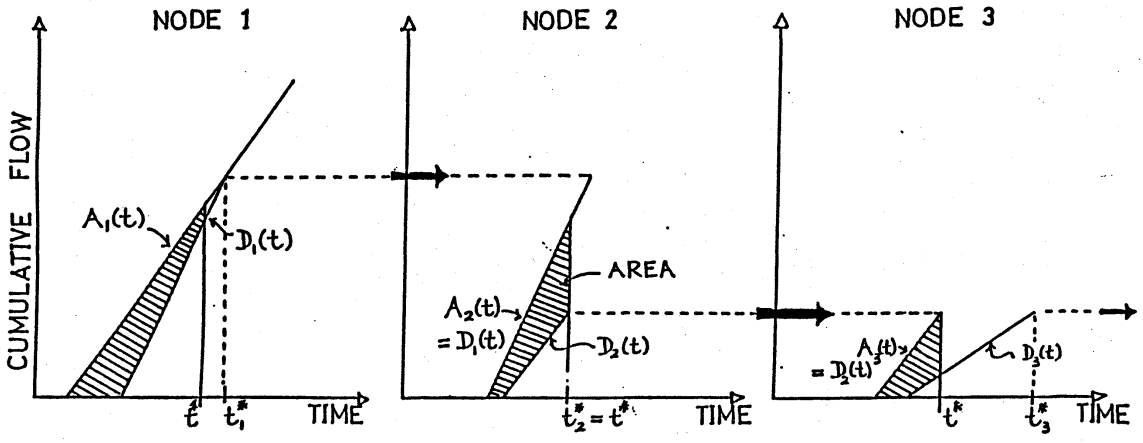

図 3　t^* の計算

たとえばノード 1 では

$$A_1(t) - D_1(t) = 0$$

のみが制約条件として課せられているならば、それをはじめて満たす点が t_1^* として求められる。すると、その時点までの $D_1(t)$ がつぎのノードに対する到着過程 $A_2(t)$ として入力され、ノード 2 の制約条件が

$$A_2(t) - D_2(t) \geqq p$$

であったとすると、これをはじめて満たす点として t_2^* が求められる。

同様に時点 t_2^* までの $D_2(t)$ がつぎのノードに対する到着過程として入力されるが、$A_3(t)$は t_2^* 以降の t に対しては一定値$A_3(t_2^*)$を保つものとして取り扱われる。ノード 3 における $D_3(t)$ が図のようであれば、時点 t_3^* で

$$A_3(t) - D_3(t) = 0$$

となる。

これら3つのノードにおける t_i^* のうちの最小値 t^* はここでは t_2^* である。そして、図の斜線部分の面積が計算される。

FFQA-GAP と FFQA-RAPID との演算時間の比較をした1例をつぎに示す。これは 2 stage のタンデム型待ち行列で第 2 の窓口における系内数が 3 を越えるときは第 1 の窓口はブロックされるという例である。第 1 、第 2 の窓口の平均サービス時間はそれぞれ 1/.51, 1/.32 とし、第 1 の窓口への平均到着率は .8 とした。第 2 の窓口における累積退去数が 50 を越えるまで計算を行った実験結果が次表の内容である。

	FFQA-RAPID	FFQA-GAP	
		δ =.5	δ =.25
ノード 1 の平均滞在時間	42.74	43.12	42.85
ノード 2 の平均滞在時間	8.28	8.22	8.26
DATA の入力時間	1分12秒	3分	3分
演算時間	2分30秒	7分	14分

これを見ると、計算結果はほとんど同じであるのに、演算時間はかなり改善されていることがうかがわれる。

5．FFQA 及び代表的なシミュレーション言語による計算結果の比較

FFQAは、いわゆる流体近似により確定的な問題として待ち行列システムを扱う計算アルゴリズムを基礎として作成されたものであるから、その演算内容はシミュレーションではない。しかし、確率論的なアプローチによって解

析的に解くことが困難な問題を主な対象としているため、流体近似によらない方法としてはシミュレーションによるのが通常であろうから、代表的なシミュレーション言語による計算結果および使用経験との比較検討をしておく必要があろう。

そこで、パソコンを利用するものとして SLAM-II、もっと大きな計算機を利用するものとして GPSS（実際に使用したのは MELCOM 用の GPDS ）を取上げ、それらを利用して同じ問題を解いた場合の計算結果や計算時間その他の使用経験を以下で述べておきたい。

また、流体近似は離散的な過程を連続的な過程に置き換えているから、連続的シミュレーションがこれに対応していると見ることも可能である。そこで、連続的シミュレーションの代表格として DYNAMO を取り上げ、パソコン上で動く B-DYNAMO を利用して得た結果とも比較する。

第１の例は鉱石運搬船の例で、艀15隻分の鉱石を積み卸すまでの平均所用時間を、艀の総数を変えながら GPDS と FFQA-RAPID とで計算し、その値を図 4 にプロットした。各ノードの平均サービス時間は、ノード 3 は１／.５であるが、その他の ノードについては、いずれも１／.８ とした。

GPDS の場合のサービス時間も一定としているので、艀の総数 9 隻のときの両者の差は離散的としたか否かの差と見てよいであろう。

第 2 の例はスリランカの銀行における待ち行列である。この銀行では、小切手を現金化する窓口業務で大量の待ちが生じている。開店と同時に 10 名近い客が押しかけ、持参した小切手を窓口に提出する。窓口にいる支払係は小切手を受け取って、それに関する必要事項をスクロール・ブックと称する帳簿に書込む。k 人分書込んだところで、それらを監査係に回す。
支払係はつぎの k 人に対して同様の仕事をする。一方、監査係は必要事項

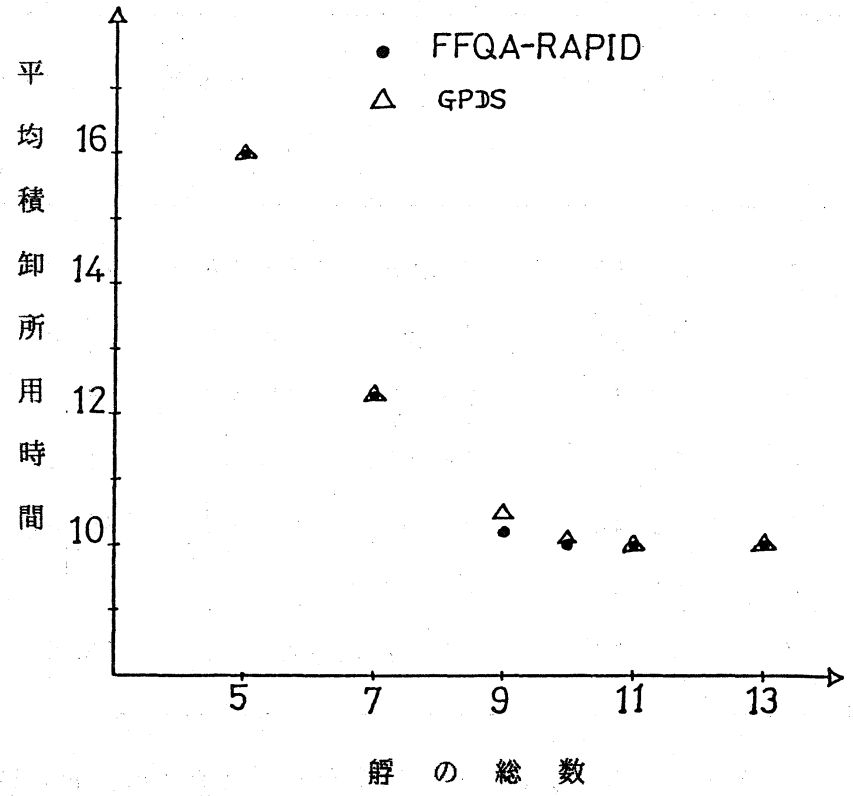

図 4 GPDS と FFQA-RAPID との比較（鉱石運搬船の例）

を検査した後、サインをして、支払係に戻す。すると、支払係は受付の業務を一旦中断して、支払い承認された小切手の現金を客に手渡す。これら受付、承認、支払の３つの仕事はそれぞれ互いに関連しているので、待ち行列システムとみれば、優先権つきフィードバックのあるタンデム型ということになり、いささか面倒になる。この問題は以前に流体近似によって取り扱ったことがあり〔1〕、その際シミュレーションを実行しているので、その結果と 今回FFQA-GAP によって計算した結果とを次表に示す。

k	FFQA-GAP	シミュレーション (FORTRAN)			
		I	II	III	IV
1	48.10	48.93	57.36	78.31	61.00
2	39.99	40.58	46.38	60.59	48.17
3	37.72	38.90	43.23	54.29	44.28
4	39.27	38.88	41.91	51.27	42.80
5	39.48	39.52	41.93	49.82	42.50
6	41.47	40.48	42.19	49.05	42.70
7	43.70	41.62	42.90	48.85	43.41

シミュレーションのケースは各サービス時間の分布の違いによる。すなわち、受付、承認、支払の３つの仕事のサービス時間の分布を

ケース I　では　すべて一定

ケース II　では　受付のみ指数分布、承認、支払は一定

ケース III　では　承認のみ指数分布、受付、支払は一定

ケース IV　では　支払のみ指数分布、受付、承認は一定

とした。

このような代表的なシミュレーション言語および FFQA の使用結果や使用経験を 10 例ほどについて比較検討した結果、流体近似の可能な場合には、平均待ち時間などを評価尺度として採用する限り、取扱いの容易さや計算時間の短さといったFFQAの実用的な優位性は、概ね確認できたと思われる。

今後、現場の諸問題への利用例が積み重ねられれば、その使用経験を踏まえて、より使いやすいプログラム・パッケージへのバージョンアップを計っていきたいと考えている。

最後に FFQA の仕様について触れておく。使用計算機は NEC製のPC-9801 (528KB)で N^{88}BASIC を利用する。プログラムの大きさは GAP が 829行、RAPID が 1131行で、8インチ または 5インチ のフロッピー 1枚で供給される。

参考文献

(1) PREMACHANDRA, I.M. and MORIMURA, H.: Modeling and Analysis of a Queueing System Existing in a Bank, JORSJ, Vol.28, pp.112-132, 1985

(2) PREMACHANDRA, I.M. and MORIMURA, H.: FFQA-RAPID AND FFQA-GAP (Manual), Research Report on Information Sciences, B-170, Tokyo Institute of Technology, 1985

(3) PREMACHANDRA, I.M. and MORIMURA, H.: A Fluid Flow Approximation Analyser for Buffer Type Queueing Systems, JORSJ, Vol.29, No.2, 1986 (印刷中)