信州大学工学部

学士論文

ポアソン到着シンボルに対して情報喪失を最小化する符号とその探索アルゴリズム

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　05T2060G
氏名　武田 修併

2009 年 2 月 28 日

目次

1　序章

1.1　研究の背景と目的

情報の伝達において, 優先される事項はそれぞれの状況に応じてさまざまである. すべての場合に共通する基準は誤り確率を最小にすることであるが, それに加えて実時間通信においては, 常に最新の情報を伝えたいという場合がある. 例えば, 遠隔地で動作している観測器があるとして, その状態を監視している状況を考えよう. この場合, 重要なのは現在の観測器の状態であり, したがって, 前の状態を誤りなく伝えることよりも現在の状態を伝えることの方が優先される. 本論文ではこのような基準を実時間基準と呼ぶことにする. 情報源にもいろいろなタイプがある. 一定の間隔でシンボルを出力するような情報源もあれば, まったく不定期にシンボルを出力するような情報源もある. 前者のタイプの情報源に対する実時間基準の符号化はシンボルの出力される間隔が符号語長を表しており本質的に一種の固定長符号化に帰着される. しかし, 後者のような情報源に対する符号化は別の問題に帰着させることができない. そこで, 従来研究 [1] ではポアソン過程に従ってシンボルを出力する情報源を考え, 実時間基準における符号化の中で, シンボルが受信者に正しく伝わらない誤り確率 ϵ ををを小さくすることを考えていた. しかし, 本論文では情報量に視点を置き, いかに情報量を多く送信できるか, つまりいかにして情報喪失を最小化するかについて考えた. まずはじめに本論文の概要について述べる.

1.2　研究の概要

実時間基準に従えば, 符号語を送信中に次のシンボルが出力されたときは, 現在の符号語の送信を中止し, 新しいシンボルに対する符号語を優先的に送信する. つまり, ひとつ前のシンボルの情報は喪失してしまうのである. そこで, 図 1 のような通信システムを考える. レート λ のポアソン過程 $N(t)$ に従うタイミングで情報源からシンボルが出力され, 符号器で符号化され, 送信レート R で送信されるモデルを考えている. そこでまず始めに, 伝わる情報量の期待値を導出していく.

2　ポアソン到着について

ポアソン到着とは, ランダムに到着するシンボルを表す確率過程のひとつである. ポアソン到着のシンボルの到着間隔は指数分布にしたがう.

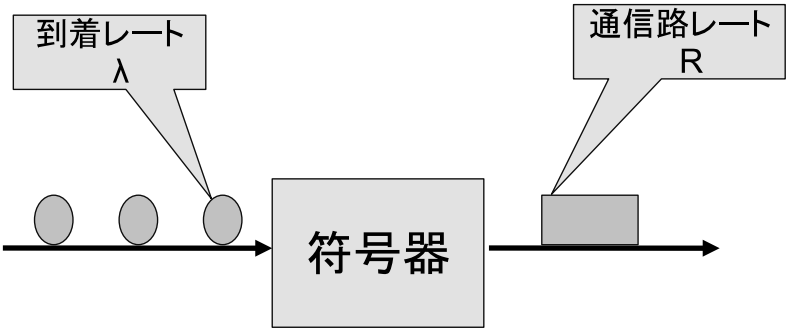

図 1　通信システムの構成

2.1　ポアソン到着の定義と基本的性質

ポアソン到着 [2][3] は, シンボルの到着が次の 3 つの条件を満たすとして仮定した 1 つの到着の仕方のモデルである. 到着間隔は指数分布に従う. ポアソン到着は以下の性質をもつ.

- 定常性
  $t > 0, x \geq 0$ なる任意の実数 t, 任意の整数 x に対し, 時間間隔 $(a, a + t)$ の間に x 人到着する確率は, 全ての a について同一で,$v_x(t)$ と表すことができる. つまり, 時間区間の幅が同じであれば, どこをとってもシステムの確率的状態は同じである. この性質を定常性という.
- 独立性
  上記に記した $v_x(t)$ は, 時刻 a までにどれだけきたか, またいつきたかには無関係である. もし前に到着したシンボルの時刻や個数によってこの確率 $v_x(t)$ が影響を受けるならば, 前のシンボルはあとに残る影響, つまり残留効果をもつことになるが, そういうことがないというのが, この独立性である.
- 希少性
  $\psi(t) = \sum_{x=2}^{\infty} v_x(t)$ と置くとき,t が十分小さければ $\psi(t)$ は無視してよい. つまり,

$$\psi(t) = o(t), (t \to 0)$$

  が成り立つ. この式は,

$$\lim_{t \to 0} \frac{\psi(t)}{t} = 0$$

を意味する．$o(t)$ とは t が十分小さければ無視してよい程度であることを示している．
つまり $\psi(t) = o(t)$ は,2 つ以上のシンボルが同時に連れだってこないことを意味する．

3 情報量の期待値

3.1 情報量の期待値の定式化

$N(t)$ は時刻 0 から t までに到着するシンボルの数を表すとする. つまり $N(t)$ は, 以下の三つの条件を満たすレート λ のポアソン過程である.

$$N(0) = 0 \tag{1}$$

$$N(t) \text{ は独立増加する.} \tag{2}$$

任意の $s, t \geq 0$ に対して

$$\Pr\{N(t+s) - N(s) = k\} = e^{-\lambda t}\frac{(\lambda t)^k}{k!} \tag{3}$$

情報源から出力されるシンボルに関して, そのタイミングはレート λ のポアソン過程 $N(t)$ に従い, シンボルの種類はポアソン過程とは独立な分布 P をもつ確率変数 X で表されるとする. シンボルは有限集合 $\mathcal{X}$ から選ばれるものとする. 通信路は単位時間あたりに長さ R の符号語を送る能力を持っているとする. ここで, 符号語の長さを $l(x)$ と表すと, シンボル $x \in \mathcal{X}$ に対する符号語を送信するのに必要な時間は

$$\frac{l(x)}{R} \tag{4}$$

で表すことができる. 実時間基準に従えば, 時刻 t に出力されるシンボル X に対する符号語を送信中に次のシンボルが出力された時は, 先に送信された符号語の送信を中止し, 新しいシンボルに対する符号語を優先的に送信する. このとき, 先に送信されていたシンボル X は受信者には正しく伝わらず喪失する. これに基づき X を送信できる条件は

$$N\left(t + \frac{l(x)}{R}\right) - N(t) = 0 \tag{5}$$

である. 喪失確率を ϵ とすると, 式 (5) と条件 (1),(2),(3) より,X を送信できる確率は

$$\begin{aligned}
1 - \epsilon &= \Pr\left\{N\left(t + \frac{l(x)}{R}\right) - N(t) = 0\right\} \\
&= \Pr\left\{N\left(\frac{l(x)}{R}\right) - N(0) = 0\right\} \\
&= \Pr\left\{N\left(\frac{l(x)}{R}\right) = 0\right\} \\
&= \exp\left(-\frac{\lambda}{R}l(x)\right)
\end{aligned} \tag{6}$$

となる. ここで X の情報量は

$$\log \frac{1}{P(x)} \tag{7}$$

で表されるので,X を受け取った元での送信できる情報量の期待値は

$$\exp\left(-\frac{\lambda}{R}l(x)\right)\log\frac{1}{P(x)} \tag{8}$$

である. よって, 伝わる情報量の期待値 I は

$$I=\sum_{x\in\mathcal{X}}\exp\left(-\frac{\lambda}{R}l(x)\right)P(x)\log\frac{1}{P(x)} \tag{9}$$

で求めることができる.

3.2 期待値 I の最大化の導出

重要な関心は, 与えられた通信路レート R とポアソン過程のレート λ に対して, 伝わる情報量の期待値 I の上限がいくらになるかである. そこで, 理想符号語長という視点から語頭符号を用いて先ほど求めた伝わる情報量の期待値 I を最大化していく. 伝わる情報量の期待値 I をクラフトの不等式の等号が成り立つ条件の下, つまり

$$\sum_{x\in\mathcal{X}}\exp\{-l(x)\}=1 \tag{10}$$

の下で I をラグランジュ未定乗数法を用いて最大化した I^* は

$$I^*=\left(\sum_{x\in\mathcal{X}}(-P(x)\log P(x))^{\frac{R}{R-\lambda}}\right)^{\frac{R-\lambda}{R}} \tag{11}$$

となり, これを達成する理想符号語長は

$$l^*(x)=\log\frac{\sum_{\tilde{x}\in\mathcal{X}}(-P(\tilde{x})\log P(\tilde{x}))^{\frac{R}{R-\lambda}}}{(-P(x)\log P(x))^{\frac{R}{R-\lambda}}} \tag{12}$$

となる. ここに対数の底は e である. 導出の詳細は付録 A に示す.

3.3 実時間基準の下でのエントロピー符号化とハフマン符号化

平均符号語長を短くするのに最適とされている $l(x)=-\log P(x)$ とラグランジュ未定乗数法で求めた理想符号語長との比較を行った. 最適符号語長 $l(x)=-\log P(x)$ については付録

B に証明を示す. いま,$\mathcal{X} = \{0, 1, 2, 3\}, P(0) = P(1) = P(2) = \frac{2}{7}, P(3) = \frac{1}{7}, \lambda = 1, R = 2$ とする. このときエントロピー符号語長を用いた伝わる情報量の期待値を $I^{\sim}$ とすると

$$I^* = 0.679424$$

$$I^{\sim} = 0.679038$$

となる. この例によって平均符号語長を短くするという目的の場合に理想とされる $l(x) = -\log P(x)$ は実時間基準においては最適ではないことを実証している. 同様にハフマン符号化を用いて比較を行う. いま,$\mathcal{X} = \{0, 1, 2, 3\}, P(0) = P(1) = P(2) = \frac{9}{29}, P(3) = \frac{2}{29}, \lambda = 1, R = 2$ とする. このときハフマン符号化すると符号語長は $(2, 2, 2, 2)$ となる. そのときの伝わる情報量の期待値を I は

$$I = 0.468605$$

となる. しかしながら符号語長 $(1, 2, 3, 3)$ を使うと伝わる情報量の期待値を I は

$$I = 0.476008$$

となる. 上の例を見るとハフマン符号を使ってない方が伝わる情報量の期待値を I が大きいことが分かる. よって実時間基準ではハフマン符号も最適とは言えないことが分かった.

4　探索アルゴリズム

4.1　基本探索アルゴリズムの作成

ラグランジュ未定乗数法で求めた符号語長は実数値で表されているが, 実用化を図るために符号語長が整数値の場合でも求めたいと考える. そのために符号語長が整数でアルファベットサイズを指定するだけで情報量の期待値の最大値を求めることができる探索アルゴリズムを考える. 符号として語頭符号を考えるので, 葉の数が指定されたアルファベットサイズに等しい木を探索すればよい.

そのモデルを図 2 に表す.

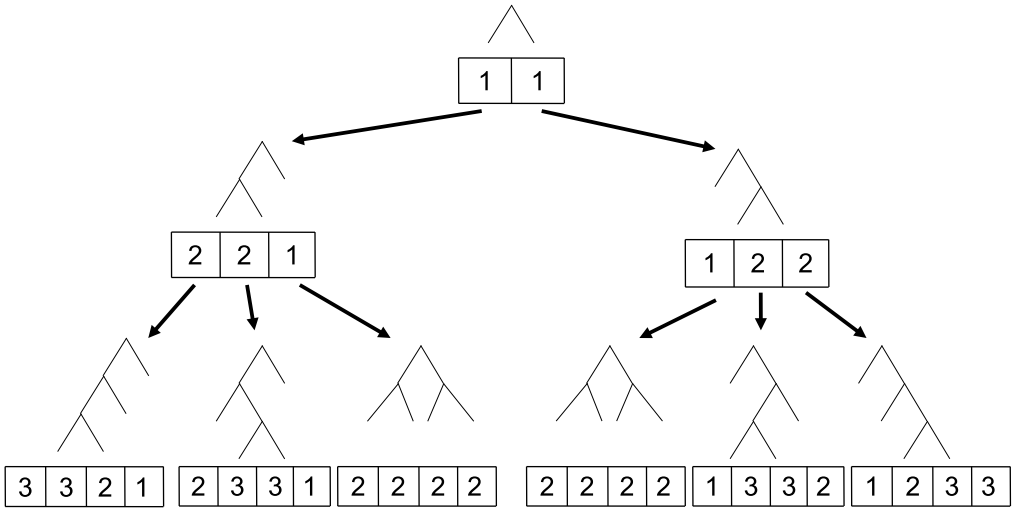

図 2　探索アルゴリズムのモデル

上図は指定されたアルファベットサイズが 4 の時を仮定したモデル図である. 始めはアルファベットサイズが 2 の木からスタートして指定されたアルファベットサイズが 4 なのでサイズ 2 の木の左の葉から枝を伸ばしてサイズ 3 の木を作る. まだ指定されたアルファベットサイズではないのでさらに一番左の葉から枝を伸ばしてサイズ 4 の木を作る. ここでようやく指定されたアルファベットサイズになったのでできた木の符号語長を式 (9) に代入して情報量の期待値を求める. そしてサイズ 3 の木に戻って今度は真ん中の葉から枝を伸ばして違う符号語長のサイズ 4 の木を作る. そして同様に情報量の期待値を求めて先ほどの期待値とどちらが大きいか比較して大きい方の値と符号語長を残しておく. これを再帰的に繰り返すことによって, アルファベットサイズ 4 の木をすべてを作り全部の情報量の期待値を計算して情報量の期待値

の最大値を求めることができる. 図 2 の木の下の数字は作成された木を符号語長で表している.

4.2 符号語長が実数と整数の比較

4.2.1 実験内容

探索アルゴリズムを使い, 符号語長が実数と整数では伝わる情報量の期待値がどのように違うのか比較する実験を行う. 探索アルゴリズムによって求められた符号語長に確率を割り当てて, 伝わる情報量の期待値 I に代入して求められた情報量の期待値とラグランジュ未定乗数法で最大化した I^* に確率を代入することによって求められた情報量の期待値を比較する. ここで確率の値はシンボルごとに $\Pr\{0\} = P, \Pr\{1\} = 1 - P$ で確率を決めてブロックに含まれる 0 と 1 の確率の積で決まる. これをアルファベット拡大して, アルファベットサイズを大きくして考えている. P の値を 0.5〜1.0 まで 0.1 刻みで変化させて測定した.

4.2.2 結果と考察

実験結果を図 3〜7 に示す. 図 3〜7 は横軸にアルファベットサイズ, 縦軸に情報量の期待値とする.

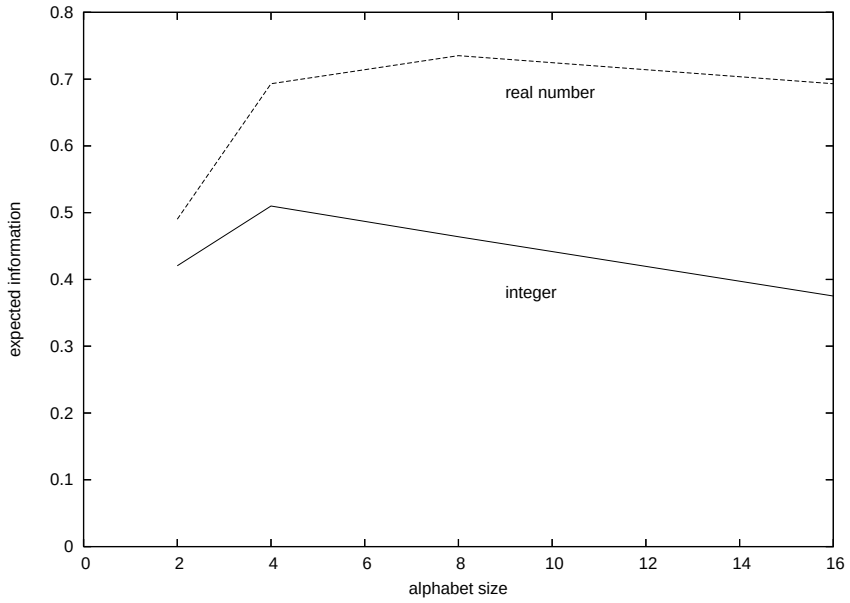

図 3　符号語長が実数と整数の比較 (確率 P=0.5)

図 3〜7 より符号語長が整数の場合は実数の値より大幅に小さくなってしまうので, 実用的

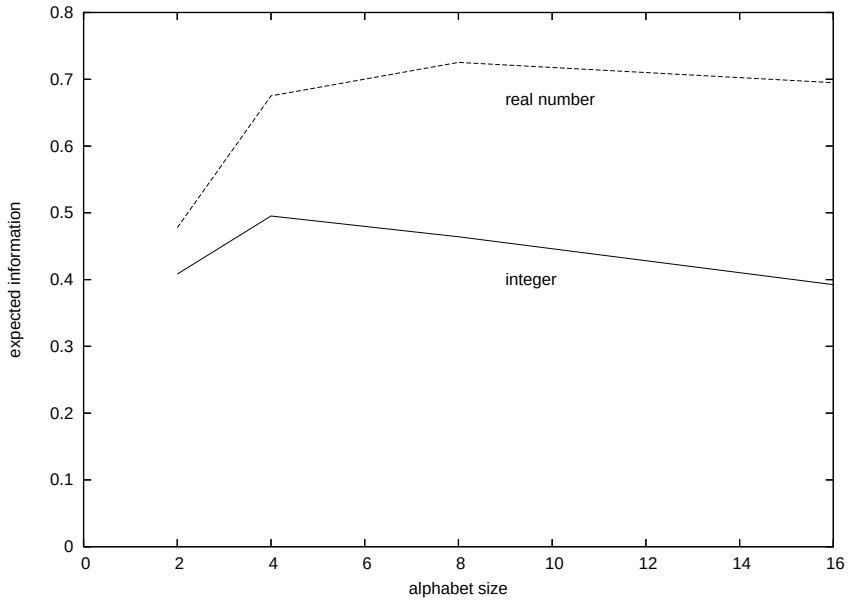

図 4　符号語長が実数と整数の比較 (確率 P=0.6)

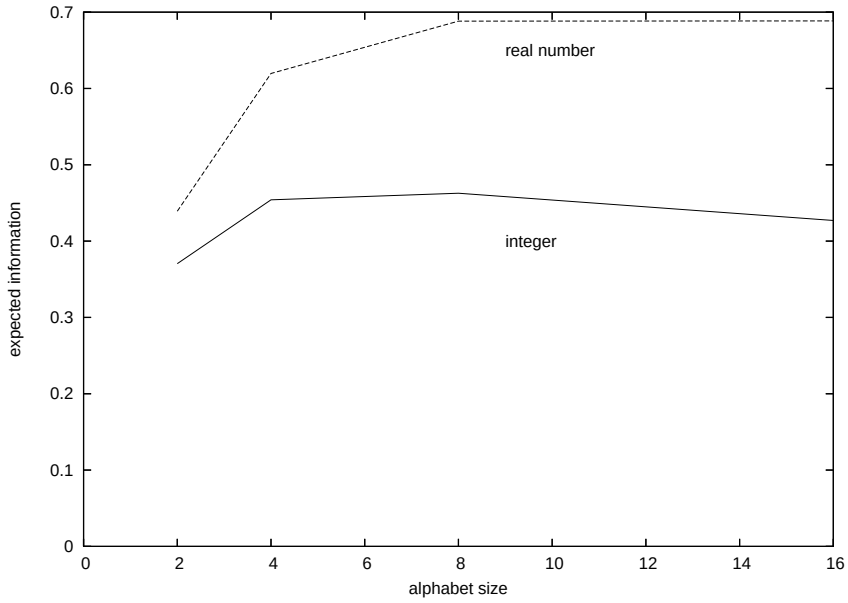

図 5　符号語長が実数と整数の比較 (確率 P=0.7)

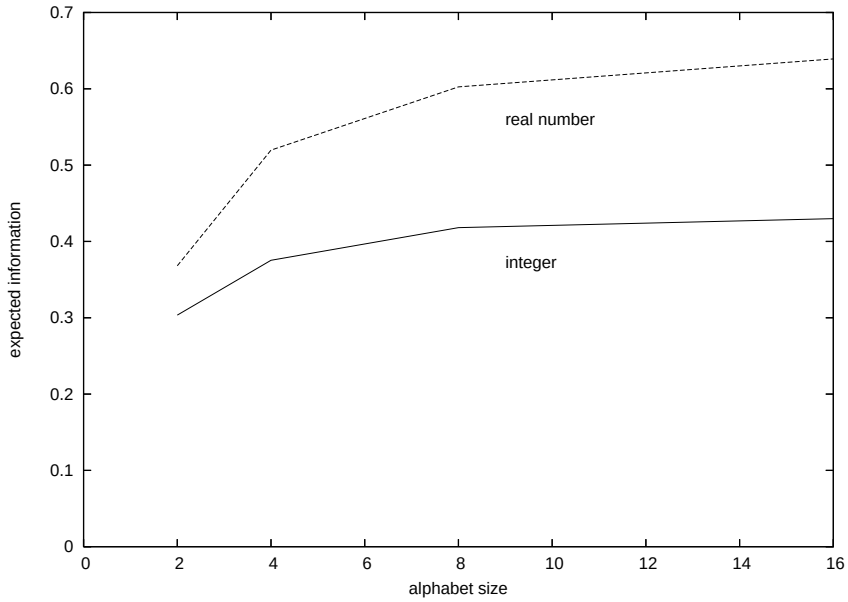

図 6　符号語長が実数と整数の比較 (確率 P=0.8)

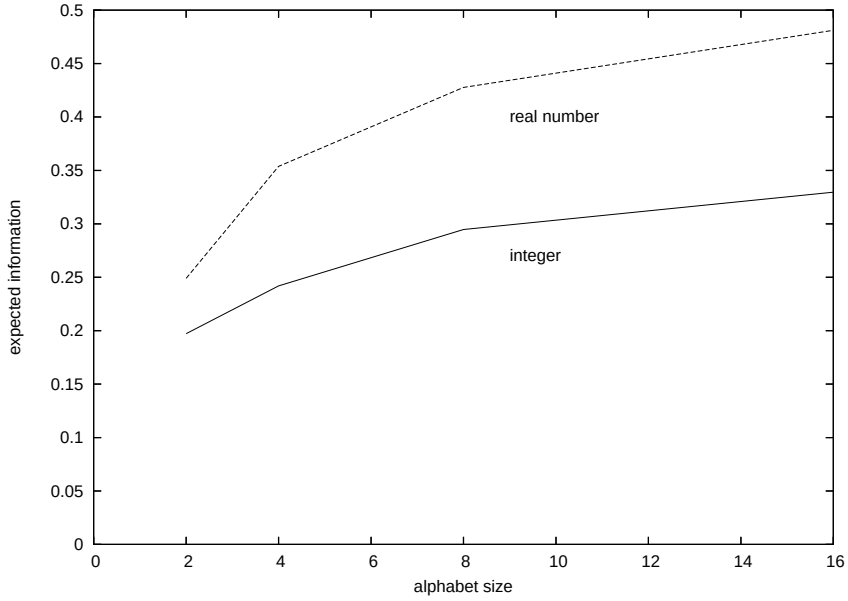

図 7　符号語長が実数と整数の比較 (確率 P=0.9)

な面からはラグランジュ未定乗数法で最大化しても参考にならないことが分かった. 従って符号語長が整数で求めるための探索アルゴリズムが重要である. また確率 P を 0.5 に近づけていくと, アルファベットサイズが大きくなるにつれて情報量の期待値が小さくなっていくことが分かる. これはアルファベットサイズが大きいと送信するための時間が長くなってしまい, 次のシンボルが到着して情報喪失が起こる確率が高くなるためであると考えた.

4.3 探索アルゴリズムの効率の改良

4.3.1 効率のいいアルゴリズムの提案

図 3〜7 を求めるために使用した探索アルゴリズムは指定されたアルファベットサイズまでの木をすべて網羅して作る愚直なアルゴリズムだった. これをアルゴリズム 1 と呼ぶことにする. 探索アルゴリズム 1 はアルファベットサイズが大きくなると作成する木の数が爆発的に増えてしまうので探索に時間がかかってしまう. そのため, 確率を大きい順に符号語に割り当てる性質を利用して作成する木の数を減らし効率を良くしようと考えた. 短い符号語長に大きい確率を割り当てることが情報量の期待値を大きくするのに最適であることは明白であるので符号語長が小さい順に並んでいない木が現れた場合はそれ以上枝を伸ばさないでその先の木を作らないことによって無駄な木の出現を減らした. この方法をアルゴリズム 2 とする. なぜアルゴリズム 2 で木を削りすぎていないかというと, 符号語長が小さい順に並んでいない木からでも枝を伸ばしていけばまた符号語長が小さい順に並んでいる木を作ることができる. しかし, それによってできた木は必ず符号語長が小さい順に並んでいる木から枝を伸ばしていけば作ることができる. なぜなら符号語長が小さい順に並んでいる木は, アルファベットサイズを一つ小さくしてもまた符号語長が小さい順に並んでいる木を作ることができるので, 反対に考えると符号語長が小さい順に並んでいる木を残しておけばアルファベットサイズを大きくしていっても符号語長が小さい順の木を網羅して作ることができる. このことから木を削りすぎていないことが証明できる.

4.3.2 実験内容

アルゴリズム 1 とアルゴリズム 2 の二つの方法による作られた木を式 (9) に代入して計算する評価回数と再帰関数のループ回数比較する実験を行い図 8,9 に示した. 図 8 は横軸をアルファベットサイズ, 縦軸を評価回数として, 図 9 は横軸をアルファベットサイズ, 縦軸をループ回数とした.

4.3.3 結果と考察

アルゴリズム 2 ではアルゴリズム 1 より作る木の数を減らすことができループ回数も減らすことができた. しかしそれでもアルファベットサイズが増えていくにつれて同じ符号語長の木

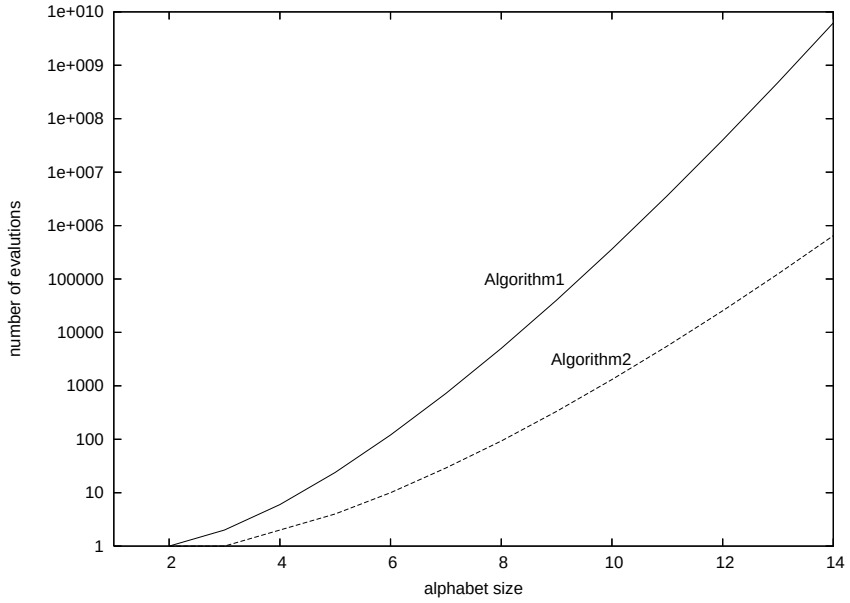

図 8　評価回数の比較

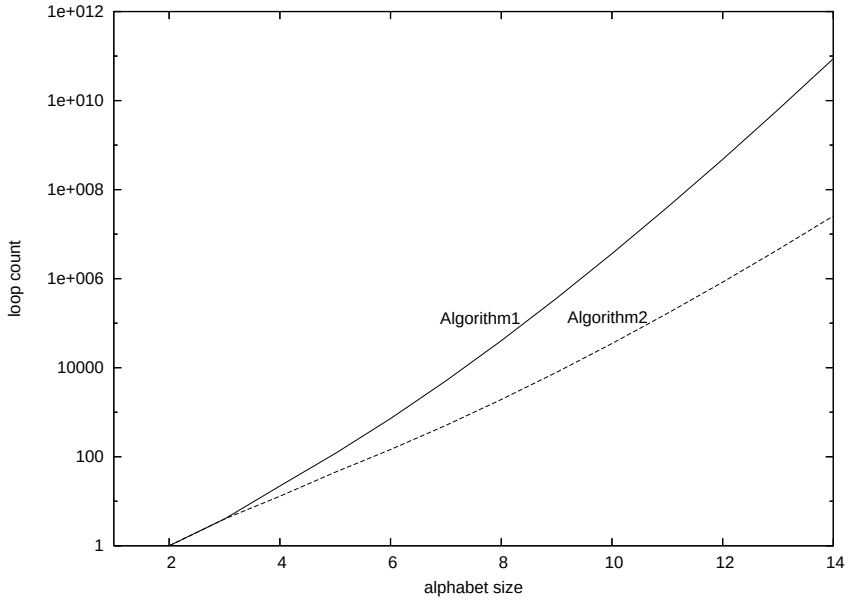

図 9　ループ回数の比較

の重複が見られた. 例としてアルゴリズム 2 の方法でアルファベットサイズ 5 までの木の作成を図 10 に示した. 図 10 を見るとアルファベットサイズ 5 の木が重複して作られていることが分かる. アルファベットサイズをさらに大きくしていくとさらに多くの重複した木が現れると考えられる. この木の重複を失くすことでさらにアルゴリズムの効率を向上できると考えた.

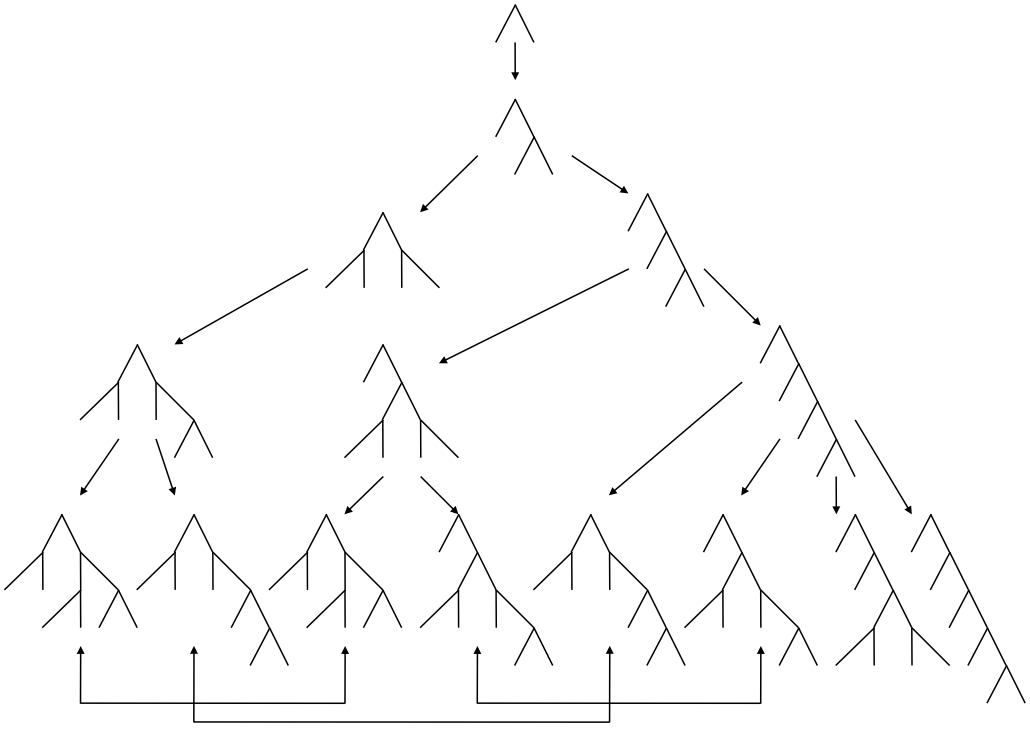

図 10　重複する符号語長の木の例

4.4　新しい探索アルゴリズムの提案

4.4.1　漸化式の提案

アルゴリズム 2 において木の重複の発生をなくすために, 符号語長が小さい順に並んでいて重複しない木の数を求めようと考える. そこで図 11 のようなモデルを考えた.N_k をアルファベットサイズが k の時の符号語長が小さい順に並んでいて重複しない木の数とするとアルファベットサイズ 1 の木は一つしかないので $N_1 = 1$, サイズ 2 の木も一つしかないので $N_2 = 1$ と表すことができる. ここでアルファベットサイズ 3 の木はサイズ 1 とサイズ 2 の木の組み合わせと考えることができる. そこでサイズ 3 の木の数を $N_3 = N_1 \times N_2 = 1$ と求めることができる. アルファベットサイズ 4 の木は符号語長が $2, 2, 2, 2$ の木はサイズ 2 の木を二つ組み合わ

せた木と考えることができる. 符号語長が $1,2,3,3$ の方はサイズ 1 とサイズ 3 の木の組み合わせと考える. このことからサイズ 4 の木の数は $N_4 = N_1 \times N_3 + N_2 \times N_2 = 1+1 = 2$ と表現できる. 一般にここで求めたいサイズ k の木は, 根の左の枝の先にサイズ i の木をつなぎ, 右の枝の先にサイズ j の木をつなぐことによってできる. ただし $i+j=k$ かつ $i \leq j$ でなければならない. この法則に従って, 符号語長が小さい順に並んでいて重複しない木の数を求める漸化式を見つけた. その漸化式は

$$N_k = \sum_{1 \leq i \leq \frac{k}{2}} N_i N_{k-i} \tag{13}$$

で表されることが分かった. この漸化式に従って探索アルゴリズム 3 を作ることができる. 式

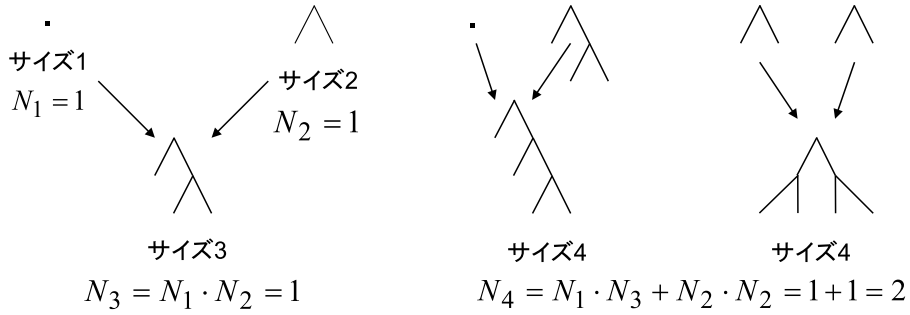

図 11　漸化式のモデル

(13) について通常の母関数や指数母関数 [4] を使って解いてみようと試みたが, 解くまでにはいたっていない.

4.4.2　実験内容

アルゴリズム 1,2,3 の三つの方法の評価回数を比較する実験を行い図 12 に示した.

4.4.3　結果と考察

図 12 より指数的な爆発はしているものの, 評価回数が大幅に減っており, 探索アルゴリズムの効率が改善できた.

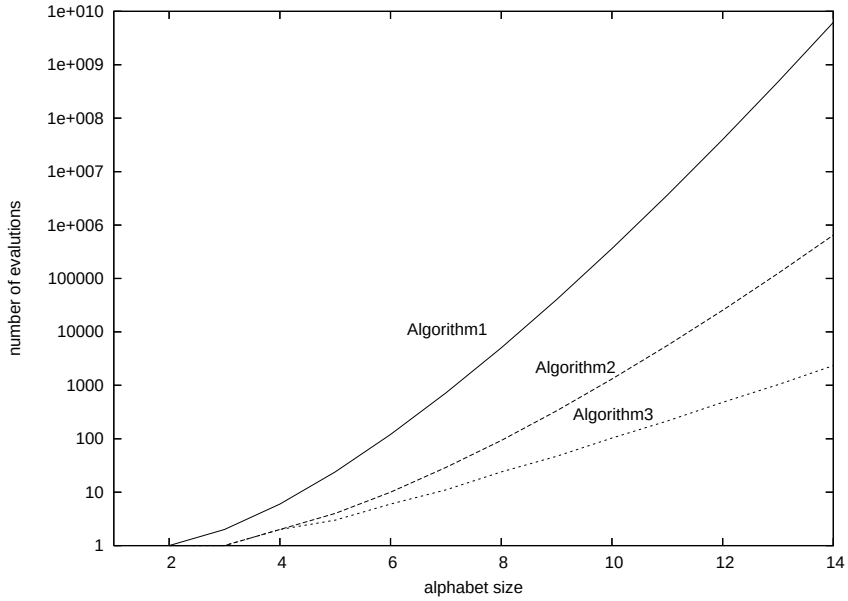

図 12　評価回数の比較

5　まとめ

本研究で用いた伝送モデルにおいての, 伝わる情報量の期待値の理論式を求めることができ, ラグランジュ未定乗数法で最大化することができた. 実時間基準においてはエントロピー符号化は最適ではないことが分かった. 同様にハフマン符号化も最適とは限らないことが分かった. ラグランジュ未定乗数法で求めた情報量の最大値は符号語長が実数であるため, 符号語長が整数の場合の最大値と大きな差があり, 実際には整数の符号語長で求めることが重要であることが分かった. 探索プログラムにおいて重複する木を作らないようにするアルゴリズムが存在することが分かった. 今後の課題としては符号語長が小さい順に並んでいて重複しない木の数を求めるための漸化式を使っての探索アルゴリズムを作成することと, 式 (13) を解くことが挙げられる.

謝辞

本研究を行うにあたって，細かく指導してくださった指導教員の西新幹彦准教授に感謝の意を表する.

参考文献

[1] M. Nishiara, “ On Coding for Sources That Output the Symbols Accoding to Poisson Process,” IEICE Trans. Fundamentals, vol.E89-A, no.10, pp.2906-2912, October 2006.

[2] 森村英典, 大前義次, 応用待ち行列理論, 日本技連, 1975.

[3] 西田俊夫, 待ち行列の理論と応用, 朝倉書店, 1971.

[4] R. L. Graham, D. E. Knuth, O. Patashnik, (有澤誠, 安村通晃, 萩野達也 訳), コンピュータの数学, 共立出版株式会社, 1993.

[5] T. M. Cover, J. A. Thomas, Elements Of Information Theory Second Edition, Wiley Interscience, 2006.

付録 A　期待値 I の最大化の導出の詳細

伝わる情報量の期待値 I

$$I = \sum_{x \in \mathcal{X}} \exp\left(-\frac{\lambda}{R} l(x)\right) P(x) \log \frac{1}{P(x)} \tag{14}$$

をクラフトの不等式の等号が成り立つ条件の下, つまり

$$\sum_{x \in \mathcal{X}} \exp\{-l(x)\} = 1 \tag{15}$$

の下で I をラグランジュ未定乗数法を用いて最大化する. ラグランジュ乗数を ξ として

$$\Phi = \sum_{x \in \mathcal{X}} \exp\left(-\frac{\lambda}{R} l(x)\right) P(x) \log \frac{1}{P(x)} - \xi \left(\sum_{x \in \mathcal{X}} e^{-l(x)} - 1 \right) \tag{16}$$

とおく. ここで $x \in \mathcal{X}$ に対して Φ を $l(x)$ で微分すると

$$\frac{\delta \Phi}{\delta l(x)} = -\exp\left(-\frac{\lambda}{R} l(x)\right) \frac{\lambda}{R} P(x) \log \frac{1}{P(x)} + \xi e^{-l(x)} \tag{17}$$

となる. これを 0 とおいて $l(x)$ について解くと

$$\exp\left(-\frac{\lambda}{R} l(x)\right) = -\frac{R}{\lambda P(x) \log P(x)} \xi e^{-l(x)} \tag{18}$$

$$\left(e^{-l(x)}\right)^{\frac{\lambda - R}{R}} = -\frac{R\xi}{\lambda P(x) \log P(x)} \tag{19}$$

$$e^{-l(x)} = \left(-\frac{R\xi}{\lambda P(x) \log P(x)}\right)^{\frac{R}{\lambda - R}} \tag{20}$$

$$-l(x) = \frac{R}{\lambda - R} \log \frac{-R\xi}{\lambda P(x) \log P(x)} \tag{21}$$

$$l(x) = -\frac{R}{\lambda - R} \log \frac{-R\xi}{\lambda P(x) \log P(x)} \tag{22}$$

を得る. これを式 (15) に代入すると

$$\sum_{x\in\mathcal{X}}\left(-\frac{R\xi}{\lambda P(x)\log P(x)}\right)^{\frac{R}{\lambda-R}}=1 \tag{23}$$

$$\left(\frac{R\xi}{\lambda}\right)^{\frac{R}{\lambda-R}}\sum_{x\in\mathcal{X}}\left(-\frac{1}{P(x)\log P(x)}\right)^{\frac{R}{\lambda-R}}=1 \tag{24}$$

$$\left(\frac{R\xi}{\lambda}\right)^{\frac{R}{\lambda-R}}=\frac{1}{\sum_{x\in\mathcal{X}}\left(-P(x)\log P(x)\right)^{-\frac{R}{\lambda-R}}} \tag{25}$$

$$\begin{aligned}\frac{R\xi}{\lambda}&=\left(\frac{1}{\sum_{x\in\mathcal{X}}\left(-P(x)\log P(x)\right)^{\frac{R}{R-\lambda}}}\right)^{\frac{\lambda-R}{R}}\\&=\left(\sum_{x\in\mathcal{X}}\left(-P(x)\log P(x)\right)^{\frac{R}{R-\lambda}}\right)^{\frac{R-\lambda}{R}}\end{aligned} \tag{26}$$

となる. 式 (26) を式 (22) に代入すると理想符号語長 $l^*(x)$ は

$$\begin{aligned}l^*(x)&=\frac{R}{R-\lambda}\log\frac{\sum_{\tilde{x}\in\mathcal{X}}(-P(\tilde{x})\log P(\tilde{x}))^{\frac{R}{R-\lambda}}}{(-P(x)\log P(x))}\\&=\log\frac{\sum_{\tilde{x}\in\mathcal{X}}(-P(\tilde{x})\log P(\tilde{x}))^{\frac{R}{R-\lambda}}}{(-P(x)\log P(x))^{\frac{R}{R-\lambda}}}\end{aligned} \tag{27}$$

と表される. さらに式 (27) を式 (14) に代入すると, 伝わる情報量の期待値の最大値 I^* は

$$\begin{aligned}I^*&=\sum_{x\in\mathcal{X}}\exp\left(-\frac{\lambda}{R}\log\frac{\sum_{\tilde{x}\in\mathcal{X}}(-P(\tilde{x})\log P(\tilde{x}))^{\frac{R}{R-\lambda}}}{(-P(x)\log P(x))^{\frac{R}{R-\lambda}}}\right)(-P(x)\log P(x))\\&=\sum_{x\in\mathcal{X}}\left(\frac{\sum_{\tilde{x}\in\mathcal{X}}(-P(\tilde{x})\log P(\tilde{x}))^{\frac{R}{R-\lambda}}}{(-P(x)\log P(x))^{\frac{R}{R-\lambda}}}\right)^{-\frac{\lambda}{R}}(-P(x)\log P(x))\\&=\sum_{x\in\mathcal{X}}\frac{(-P(x)\log P(x))^{\frac{\lambda}{R-\lambda}+1}}{\left(\sum_{\tilde{x}\in\mathcal{X}}(-P(\tilde{x})\log P(\tilde{x}))^{\frac{R}{R-\lambda}}\right)^{\frac{\lambda}{R}}}\\&=\left(\sum_{x\in\mathcal{X}}(-P(x)\log P(x))^{\frac{R}{R-\lambda}}\right)^{1-\frac{\lambda}{R}}\\&=\left(\sum_{x\in\mathcal{X}}(-P(x)\log P(x))^{\frac{R}{R-\lambda}}\right)^{\frac{R-\lambda}{R}}\end{aligned} \tag{28}$$

と求められる.

付録 B　最適符号語長の証明

語頭符号のクラフトの不等式が等式が成り立つ条件の元での最小符号語長 [5] を求める.

$$L = \sum_{i=1}^{n} P_i l_i \tag{29}$$

ですべての $l_1, l_2, \ldots, l_m$ がクラフトの不等式

$$\sum_{i=1}^{n} D^{-l_i} \leq 1 \tag{30}$$

を満たしているとする. ここでラグランジュ乗数法を使って最小化する. ラグランジュ乗数を λ として

$$J = \sum_{i=1}^{n} P_i l_i + \lambda \left(\sum_{i=1}^{n} D^{-l_i} \right) \tag{31}$$

とおく.J を l_i で微分すると

$$\frac{\delta J}{\delta l_i} = P_i + \lambda D^{-l_i} \log_e D \tag{32}$$

となる. これを 0 とおいて D^{-l_i} について解くと

$$P_i + \lambda D^{-l_i} \log_e D = 0 \tag{33}$$

$$D^{-l_i} = \frac{P_i}{\lambda \log_e D} \tag{34}$$

を得る. これを条件の $\sum_{i=1}^{n} D^{-l_i} = 1$ に代入すると

$$\sum_{i=1}^{n} \frac{P_i}{\lambda \log_e D} = 1 \tag{35}$$

$$\lambda = \frac{1}{\log_e D} \tag{36}$$

となる. 式 (19) に代入すると

$$\begin{aligned} D^{-l_i} &= \frac{P_i \log_e D}{\log_e D} \\ &= P_i \end{aligned} \tag{37}$$

よって最小を達成する l_i は

$$l_i^* = -\log_D P_i \tag{38}$$

と求めることができる.

付録 C　ソースコード

C.1　アルゴリズム 1

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

int Leaf;
double* Prob;
double Lambda;
double R;
double ResultValue;
int* ResultTree;

unsigned long C_eval;
unsigned long C_loop;

void
calc(int* tree, int size)
{
    int i;
    double NowResultValue = 0;

    for (i = 0; i < size; i++){
        NowResultValue += -1.0 * exp(-1.0 * Lambda * tree[i] / R) * Prob[i] * log(Prob[i]);
    }
    /* printf("[%d] %f\n", __LINE__, NowResultValue); */

    if (NowResultValue > ResultValue){
        memcpy(ResultTree, tree, Leaf * sizeof(int));
        ResultValue = NowResultValue;
    }
    C_eval++;
    return;
}

void
MakeTree(int* tree, int size)
{
    int parent;
    int i;
    int j;
    int *newtree;

    if(size == Leaf){
    calc(tree, size);
    return;
    }

    for (parent = 0; parent < size; parent++){
        newtree = tree + size;
        for (i = 0, j = 0; i < size; i++){
            if (i == parent){
                newtree[j++] = tree[i] + 1;
```

```
                newtree[j++] = tree[i] + 1;
            } else {
                newtree[j++] = tree[i];
            }
            C_loop++;
        }
        MakeTree(tree + size, size + 1);
    }
    return;
}

int main(int argc, char **argv)
{
    clock_t start,end;
    int* Buffer;
    int i;

    if (argc != 3){
        printf("パラメータの数が正しくありません.in %d\n",__LINE__);
        exit(1);
    }
    Lambda = atof(argv[1]);
    R = atof(argv[2]);
    if (Lambda == 0 || R == 0){
        printf("error in %d\n", __LINE__);
        exit(1);
    }

    if (scanf("%d", &Leaf) != 1){ /* number of leaves */
        printf("error in %d\n", __LINE__);
        exit(1);
    }

    Buffer = (int*)malloc(((Leaf * (Leaf + 1) / 2) - 1) * sizeof(int));
    ResultTree = (int*)malloc(Leaf * sizeof(int));
    Prob = (double*)malloc(Leaf * sizeof(double));

    for(i = 0; i < Leaf; i++){
        if (scanf("%lf", Prob + i) != 1){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }

    start = clock();

    ResultValue = 0;
    memset(Buffer, 0, ((Leaf * (Leaf + 1) / 2) - 1) * sizeof(int));
    memset(ResultTree, 0, Leaf * sizeof(int));

    Buffer[0] = 1;
    Buffer[1] = 1;

    C_eval = 0;
    C_loop = 0;

    MakeTree(Buffer, 2);

    printf("\nResultValue=%06lf\n", ResultValue);
    printf("ResultTree: ");
    for(i = 0; i < Leaf; i++){
        printf("%d, ", ResultTree[i]);
    }
    printf("\n");

    printf("eval: %ld, loop: %ld\n", C_eval, C_loop);

    end = clock();
```

```
    printf( "処理時間:%d[ms]\n", end - start );

    return(0);
}
```

C.2 アルゴリズム 2

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

int Leaf;
double* Prob;
double Lambda;
double R;
double ResultValue;
int* ResultTree;

unsigned long C_eval;
unsigned long C_loop;

void
calc(int* tree, int size)
{
    int i;
    double NowResultValue = 0;

    for (i = 0; i < size; i++){
        NowResultValue += -1.0 * exp(-1.0 * Lambda * tree[i] / R) * Prob[i] * log(Prob[i]);
    }
    /*printf("[%d] %f\n", __LINE__, NowResultValue);*/

    if (NowResultValue > ResultValue){
        memcpy(ResultTree, tree, Leaf * sizeof(int));
        ResultValue = NowResultValue;
    }
    C_eval++;
    return;
}

void
MakeTree(int* tree, int size)
{
    int parent;
    int i;
    int j;
    int *newtree;

    if(size == Leaf){
        calc(tree, size);
        return;
    }

    for (parent = 0; parent < size; parent++){
        newtree = tree + size;
        for (i = 0, j = 0; i < size; i++){
            if (i == parent){
                newtree[j++] = tree[i] + 1;
                newtree[j++] = tree[i] + 1;
            } else {
                newtree[j++] = tree[i];
            }
```

```
            C_loop++;
        }
        for (i = 0; i<size + 1; i++){
            if(newtree[i] > newtree[i+1]) break;
        }
        if(i==size){
            MakeTree(tree+size,size+1);
        }
    }
    return;
}

int main(int argc, char **argv)
{
    int* Buffer;
    int i;
    clock_t start, end;

    if (argc != 3){
        printf("パラメータの数が正しくありません.in %d\n",__LINE__);
        exit(1);
    }
    Lambda = atof(argv[1]);
    R = atof(argv[2]);
    if (Lambda == 0 || R == 0){
        printf("error in %d\n", __LINE__);
        exit(1);
    }


    if (scanf("%d", &Leaf) != 1){ /* number of leaves */
        printf("error in %d\n", __LINE__);
        exit(1);
    }

    Buffer = (int*)malloc(((Leaf * (Leaf + 1) / 2) - 1) * sizeof(int));
    ResultTree = (int*)malloc(Leaf * sizeof(int));
    Prob = (double*)malloc(Leaf * sizeof(double));

    for(i = 0; i < Leaf; i++){
        if (scanf("%lf", Prob + i) != 1){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }

    start = clock();

    ResultValue = 0;
    memset(Buffer, 0, ((Leaf * (Leaf + 1) / 2) - 1) * sizeof(int));
    memset(ResultTree, 0, Leaf * sizeof(int));

    Buffer[0] = 1;
    Buffer[1] = 1;

    C_eval = 0;
    C_loop = 0;

    MakeTree(Buffer, 2);

    printf("\nResultValue=%06lf\n", ResultValue);
    printf("ResultTree: ");
    for(i = 0; i < Leaf; i++){
        printf("%d, ", ResultTree[i]);
    }
    printf("\n");

    printf("eval: %ld, loop: %ld\n", C_eval, C_loop);
```

```
    end = clock();

    printf( "処理時間:%d[ms]\n", end - start );

    return(0);
}
```

C.3 アルゴリズム 3

```
#include <stdio.h>

int function(int k)
{
    int r, i;
    r=0;

    switch (k)
    {
        case 1:
        case 2:
        case 3:
            return 1;
    }

    for(i=1; i<=(k/2); i++)
    {
        r+=function(i)*function(k-i);
    }

    return r;
}

int main()
{
    int input;

    printf("k=");
    scanf("%d", &input);

    printf("Result=%d\n", function(input));

    return 0;
}
```