2008－HPC－116 (7)
2008／8／5

統計的パターン認識手法による ソフトウェア自動チューニングのための実験計画

小谷和正[†] 須田礼仁[†,††]

本稿では、ソフトウェア自動チューニングにおける実験時間を緩和するための汎用的な実験計画的手法として、統計的パターン認識手法を用いたアプローチの提案を行う。提案するアプローチは実験結果を成功と失敗に分類し、既存の結果、つまりパラメタ組とその分類の対応による学習から、成功する確率が高い実験のみを実行する。分類モデルとしては単純ベイズ (階層ベイズ) 識別器を採用した。これは変数群の独立性を仮定するが、多くの場合に期待よりもよく働くことが知られている。実験には小武守らの開発した疎行列線形方程式の反復解法ライブラリである Lis を用いた。実験で設定したパラメタ組の候補は固定で全 7035 組あり、例えばある行列については全て実行すると 1 日程度必要となるところ約 8 時間で、成功する実験のうちおよそ 9 割を実行することができた。

Experimental Design Approach for Automatic Tuning of Softwares Using Statistical Pattern Recognition Techniques

KAZUMASA KOTANI [†] and REIJI SUDA [†,††]

In this paper, as a general technique of experimental design for alleviating experimental time cost in automatic performance tuning of softwares, we propose an approach using statistical pattern recognition techniques. Our approach classfies experimental results into successes and failures, then learns from existing results, pairs of a parameter configuration and its class, and executes the candidate patterns which are predicted to succeed.

As a classification model, we used Naïve Bayes Classifier. Even with strong independence assumptions of parameter variables, this classifier is known to work well in many applications.

In the experiments, Lis (H. Kotakemori et al. 2005), the library of iterative solvers for linear systems, is used. With the fixed conditions set in this paper, the number of whole execution patterns becomes 7035, and one of the example matrices needs about 1 day for the entire execution. By this approach, that was reduces to about 8 hours, and the number of successful executions were around 90% of the original.

1. はじめに

計算機ハードウェアやソフトウェアの多様化に伴い、汎用的なソフトウェア自動チューニング技術に関する研究が進みつつある。しかし従来の自動チューニングソフトウェアでは、性能パラメータの組合せ爆発によるインストール時間の増大が問題となっており、より低コストでチューニングを行う手法が求められている。

性能パラメタに関する情報を効率的に、つまりより少ない実験で求める手法を我々は**実験計画**的手法と呼んでいるが、最適な性能パラメタを求める一つのアプローチとしては、実行時間などの性能値をコスト関数として、少ない標本点の実行結果からモデル推定を行うものが既に研究されている。例えば田中ら[1]によって d-スプラインを用いた逐次サンプリング法が提案されており、また我々の研究[2]では直交表や直交多項式を用いた方針の検討を行った。

本稿ではそれらの手法とは異なった新しいアプローチとして、文書分類などで広く用いられている統計的パターン認識の技術を用いた手法を提案する。ここでは、反復解法の収束性のように実行結果を何らかの意味で成功と失敗に分類できる場合を考える。例えば速度についてのデータをとりたい場合でも、そもそも収束しなくなってしまうようなパラメタでの実験はなるべく行わず成功するパターンのみを実行したい、ということがあり得る。このような場合に、既に行った実験の結果から学習を行うことで、未実行の性能パラメタ組に対してそれが成功するかどうかを推測し、成功

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo
†† JST, CREST

するであろうパターンのみ実行すれば良い、というのが基本的な考え方である。

分類モデルには単純ベイズ識別器を採用した。実験には小武守らの開発した疎行列線形方程式の反復解法ライブラリである Lis[3),4)] を用い、解法と前処理の組み合わせと、各アルゴリズムのパラメタについてのチューニングを行った。実験した行列の多くについては提案アプローチはうまく働いているようであった。

2. 手　　法

ベイズ識別器については統計的パターン認識技術として既に多くの文献が存在し、テキスト分類など応用も広く知られているものであり、ここでは簡潔に説明する。統計的パターン認識全般については文献[5)] や、詳しくは文献[6)] などを参照のこと。

提案手法は、実行済パラメタ設定とその結果の組を訓練集合として学習を行い、未実行の設定について識別モデルから成功か失敗かを推測することで、成功すると思われる候補だけをなるべく実行するというものである。

2.1 ベイズ識別器

単純ベイズ識別器は、各々の変数が独立であるという仮定を置くことで、次のような条件つき確率モデルで表現される。

$$p(C|F_1,...,F_n) = \frac{p(C)\prod_i p(F_i|C)}{p(F_1,...,F_n)}. \quad (1)$$

C は成功または失敗のクラス、F_i は特徴変数となる各性能パラメタ変数を表す。$p(C)$ は事前確率で、(成功数/実行数) とすればよいが、平滑化のために Lidstone 法を持ちいて

$$p(\text{成功}) = \frac{\text{成功数} + M}{\text{実行数} + 2M}, \quad (2)$$

とする。なお本稿においては平滑化パラメータ M は $M = 4$ とした。

また今の場合は 2 クラスであり成功と失敗の確率比を考えればよく、成否確率比

$$R(F_1,...,F_n) = \frac{p(\text{成功})}{p(\text{失敗})}\prod_i \frac{p(F_i|\text{ 成功})}{p(F_i|\text{ 失敗})}. \quad (3)$$

が、境界値 b を用いて

$$R(F_1,...,F_n) \geq b \quad (4)$$

を満たす場合は成功に分類、そうでない場合は失敗に分類する。境界値は最も単純には $b = 1$ であるが、訓練集合の少なさや偏りなどを考慮すれば、何らかの推定により適切に設定されるのが望ましい。

$p(F_i|C)$ は各パラメタ値 v_i についての尤度として、($F_i = v_i$での成功数/実行数) とすればよい。ただし本稿で例えば解法が SOR 法である場合のみその緩和係数 ω が意味をもつように、パラメタに階層構造が存在する場合がある。これは (使われていない変数は *null* に設定されているとすると)、平滑化もあわせて

$$p(F_i = v_i|\text{ 成功}) = \frac{F_i = v_i\text{での成功数} + M}{F_i \neq null\text{ での成功数} + 2M}. \quad (5)$$

と計算することによって階層構造が表現できる (階層ベイズ)。

2.2 提案手法

本稿では仮定として、パラメタ値の候補は本来連続的な値をとり得るものも含めて固定されたいくつかの値にあらかじめ定められているとする。

学習のための訓練データ集合は空の状態から始まり、実験の進行に依って増えてゆくことになるため、実験の順序は識別モデルの能力に強く影響するはずである。ここでは以下の二種類の実行順序を考える。

- 候補から無作為に選んで実行
- 成功する確率がより高いものを選んで実行

前者では次に実行する実験パターン (群) を選んだ後にそのパターンについてのみモデルによる識別を行い、失敗と分類された場合は実行をスキップすることになる。なおここでは一度スキップしたパターンについては再びチェックすることはないが、実際には繰り返し識別を行ってもよいだろう。

後者の実行順序においては、選択ステップ毎に全ての未実行パターンについて式 (3) により確率比の計算を行い、その確率比が大きいパターンを選択する。このとき、以下のように総パターン数の 2 乗オーダーで確率計算回数が増加してしまうが、しかし本稿では計算時間を緩和するために、1 ステップ毎に確率比の大きいものから 10 パターンずつを選択するのみにとどまっている。総パターン数 N、ステップ回数を M、ステップ毎の選択個数を c とすると、確率計算回数は

$$N + (N-c) + \cdots + (N-(M-1)c) = \left\{N - \frac{(M-1)c}{2}\right\}M$$

となり、$M \leq N/c$ であるからこれは $O(N^2)$ で、選択個数 c が N に依存せず定数であるならばオーダーは変わらない。確率比が厳密に最大であることは必ずしも要求されないので、より効率のいいアルゴリズムが望ましい。

3. 実験条件

実験には小武守らの開発した疎行列線形方程式の解法ライブラリ Lis[3),4)] を用いた。実験時点のバージョンでは 20 の反復解法と 11 の前処理法が実装されており、それらの組み合わせと各アルゴリズム固有のパ

ラメタは 4 値ずつを (Lis のデフォルト値を含むように) 定め、設定パターンの候補とした。値の候補は表 1 に示すが、全部で 7035 通りのパターンとなった。

行列データは "The University of Florida Sparse Matrix Collection"[7] から 14 つ選んだものを用いた。右辺ベクトルは $\mathbf{b} = (1.0, ..., 1.0)^T$ とし、初期ベクトル $x_0 = \mathbf{0}$ から、反復終了条件はアルゴリズムで算出される残差について $\|\mathbf{r}_i\|/\|\mathbf{r}_0\| \leq 1.0 * 10^{-12}$ とした。また、最大反復回数は 5000 に固定した。以下の実験では、この条件で収束した上で、さらに真の相対残差が $\|\hat{\mathbf{r}}\|/\|\mathbf{r}_0\| \leq 1.0 * 10^{-8}$ を満たしている場合にそのパラメタ組による実行は" 成功"、それ以外の場合を" 失敗" とみなした。

以下の実験環境で実際に全てのパターンについて実行した結果を表 2 に示す。それぞれのアルゴリズムによる収束性は、確かに行列に強く依存していることがわかる。

- CPU: Intel PentiumM 1.8 GHz
- L2 Cache: 2048 KBytes
- Main Memory: 1024MBytes
- OS: Debian GNU/Linux 3.3.5
- Compiler: gcc-4.2.0
- Library: Lis-1.1.0
- Optimization: -O2

4. シミュレーション実験

提案手法についての実験は、前節のように得られたデータを用いたシミュレーションによって行う。本稿では基本的に収束性のみに着目しているので、問題はないと考えられる。

4.1 実 験 1

まず、ランダムな実行順序選択について実験を行った。境界値 b は $b \in \{0.3, 0.5, 0.8, 1.0\}$ とそれぞれ固定した上で、順序をシャッフルして 20 回分の平均をとった結果を、いくつかの行列について表 3 に示す。なお、スキップしたパターンについては実験結果を学習に含めない。

最後の例以外は、本来の成功数の 9 割程度が達成されており、また成功率そのものも 9 割前後であるから、実験の効率は良いと言って良いだろう。b については、小さい値になるほど実験数と成功数が増える傾向にあるが、どの程度増減するかは行列に依って異なるようである。

最後の例については、小さい値の境界値 b では成功数は十分なものの必要な実験数の割合が他の行列に比べると多くなってしまい、逆に大きい b で実験数に伴って成功数もかなり少なくなってしまっている。これは、実行結果が単純ベイズ識別器ではあまりうまくモデル化できない挙動を示していると考えられる。

表 3 ランダム実行順序でのシミュレーション結果

行列	b	実験数	成功数	所要日数
Pres_Poisson	0.3	3097	1960	2.20
	0.5	2752	1913	1.59
	0.8	2397	1814	1.17
	1.0	2225	1744	1.00
crystm02	0.3	3493	2936	0.35
	0.5	3228	2894	0.31
	0.8	3043	2835	0.26
	1.0	2892	2742	0.22
adder_dcop_03	0.3	319	0	0.067
	0.5	163	0	0.034
	0.8	3	0	0.00065
	1.0	1	0	0.00018
c-37	0.3	778	206	0.26
	0.5	571	206	0.18
	0.8	415	201	0.080
	1.0	317	193	0.069
Na5	0.3	5820	4761	1.35
	0.5	5612	4717	1.14
	0.8	5293	4584	0.94
	1.0	5183	4541	0.87
bcsstk14	0.3	5021	2806	0.18
	0.5	4313	2632	0.15
	0.8	2754	1844	0.084
	1.0	1571	1126	0.045

4.2 実 験 2

次に、成功確率が高いものから順番に 10 個ずつ選んでいく場合について実験を行う。なお初期は全てのパターンが同様に成功率 0.5 なので、最初の 10 パターンはランダムに選んだものである。

まず、境界値は設定せずに順序付だけ行った上で全て実行したと考えた場合の結果を、実行数に対する成功数の様子として図 1 に示す。実線が実行経過、破線が全成功数、対角線が成功率 100%の線である。また、グラフ上にはランダム実行順序の場合 ($M = 2, 4, 8$、$b = 0.3, 0.5, 0.8, 1.0$) における結果も点でプロットした。実線の実行経過が対角線に近いほど、効率が良いと考えればよいだろう。

やはり前節と同じく最後の行列についてはあまりうまくいかないようだが、それ以外の行列については概ね良い結果が得られた。

また実験効率についての具体的な指標は今後の課題となるが、実験数と成功率の間にはトレードオフが発生することは確かなようである。このことは前節の結果からも同様であり、さらにこれらのグラフを見ると、二つの手法は本質的には同じ程度の効率になるということが予想される。

表 1　性能パラメタ変数ごとに定めた候補値

Solver	CG, BiCG, CGS, BiCGSTAB, BiCGSTAB(l), GPBiCG, TFQMR, Orthomin(m), GMRES(m), Jacobi, Gauss-Seidel, SOR, BiCGSafe, CR, BiCR, CRS, BiCRSTAB, GPBiCR, BiCRSafe, FGMRES(m)
Preconditioner	none, Jacobi, ILU(k), SSOR, Hybrid, I+S, SAINV, SA-AMG, Crout ILU, ILUT, additive schwarz
Scaling	none, Jacobi, Diagonal
BiCGSTAB_l	2,3,4,5
Orthomin_restart	20,40,60,80
GMRES_restart	20,40,60,80
SOR_W	1.8, 1.85, 1.9, 1.95
FGMRES_restart	20,40,60,80
ilu_fill	0, 1,2, 3
ssor_w	0.8, 1.0, 1.2, 1.4
is_alpha	1.0, 1.5, 2.0, 2.5
is_m	2, 3, 4, 5
sainv_drop	0.05, 0.01, 0.005, 0.001
saamg_unsym	true, false
iluc_drop	0.05, 0.01, 0.005, 0.001
iluc_rate	3.0, 4.0, 5.0, 6.0
ilut_drop	0.05, 0.01, 0.005, 0.001
ilut_rate	3.0, 4.0, 5.0, 6.0
adds_iter	1, 2

表 2　各行列についての総実行時間と総時間

行列名	行列サイズ	実行時間 (日)	成功数	最速パターン
Grund/poli_large	15,575	0.24	5745	bicgstab+iluc
LiuWenzhuo/powersim	15,838	1.79	2630	crs+iluc
ACUSIM/Pres_Poisson	14,822	10.5	2019	cg+iluc
Boeing/crystm02	13,965	1.05	3004	cg+iluc
FEMLAB/poisson3Da	13,514	1.61	5673	bicgstab+ssor
Pajek/foldoc	13,356	1.68	0	–
Nemeth/nemeth12	9,506	0.49	6277	cg+jacobi
Schenk_IBMNA/c-37	8,204	2.41	215	cg+ilu
Mathworks/Pd	8,081	0.73	2055	bicgstab+iluc
PARSEC/Na5	5,832	1.98	4775	cg+iluc
Grund/meg4	5,806	1.31	676	crs+iluc
Hamm/add32	4,960	0.21	5891	cg+iluc
Sandia/adder_dcop_03	1,813	1.50	0	–
HB/bcsstk14	1,806	0.40	2936	fgmres+iluc

4.3　終了条件の設定

境界値のパラメタは、失敗パターンの実行をどれだけ許容することでモデルでうまく表現できない部分をカバーするかというトレードオフを定める意味を持っていると考えられる。しかし真の成功数は実際には不明であり、何らかの推定をしなければそのトレードオフを制御できない。

ここでは実験 2 について、以下の 3 通りの方法で、未実行パターン中の成功数を推定することを考える。

(1)　未実行パターンの成功率の和
(2)　未実行パターン数 × 最大成功率
(3)　境界 b を定めて実際に分類した成功数

なお 3 つめの推定では、「残り成功数が 0 という条件」が実験の経過において「未実行パターンの成否確率が b を超えない」という条件と同値になっている。

ある行列について、上記の推定値と実際の残り成功数の様子を図 2 に示す。

確率の値そのものを加算した二つはあまりよくない結果のようである。これは、実験結果が成功に偏っていることと、モデルから算出される確率値に対する失敗一つあたりの影響が小さくなってしまうことに起因するものと思われる。分類による成功数の推定は他の二つよりは良いが、境界値パラメタ b の決定は問題である。

4.4　モデルの評価

モデルの能力を評価する場合、リサンプリング法[5),6)]が用いられる。これは、既存のデータを訓練集合と評価用集合に用いて、識別の正答率を評価する方法である。リサンプリング法には、既存データを全て訓練集合かつ評価用集合に用いる再代入法などいくつかの種類があるが、実験では一点除外法を用いた。

正答率は境界値に依存する。ある行列について、全

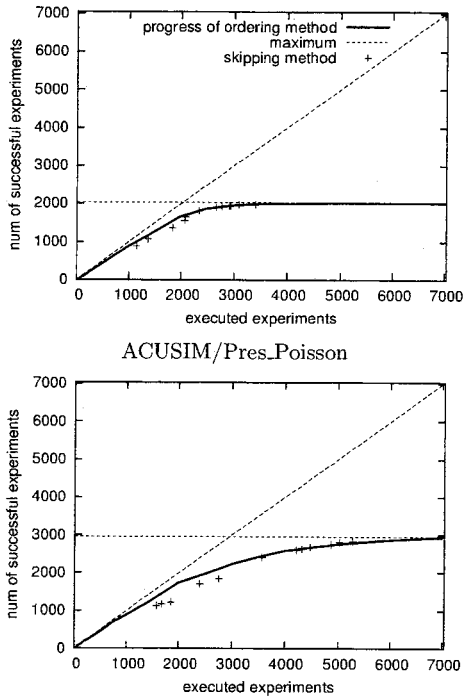


図 1　高確率パターンを優先して実行する場合の経過

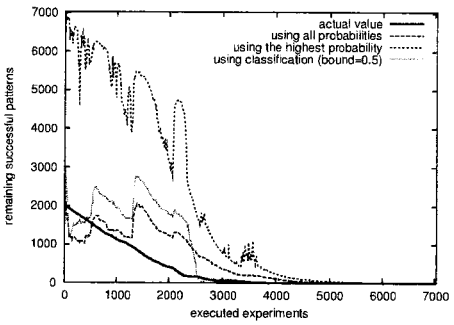


図 2　残り成功数の推定 (Pres_Poisson)

体での正答率、本来成功するものを成功と分類した割合 (correctly accept)、失敗するものを失敗と分類した割合 (correctly reject) の 3 つをプロットしたものを例として図 3 に示す。これは $b = 1.0$ と $b = 0.1$ のグラフであるが、正答率は大きく異なる。

本来は成否確率比の最大値 (全て失敗に分類) から最小値 (全て成功に分類) までを厳密に探索すべきかもしれないが、ここでは $b \in \{0.050, 0.10, 0.50, 1.0, 2.0, 5.0, 10\}$ の値から、全体での正答率が最大になるものを各ステップで選択した結果を図 4 に、その値での正答率を図 5 に示す。

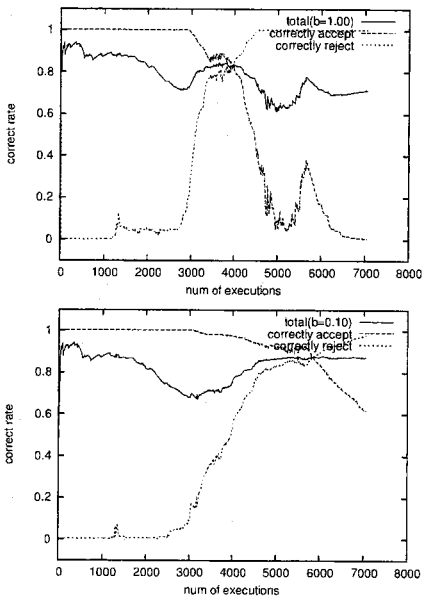


図 3　正答率の評価 (Pres_Poisson) b=1.0 or 0.1

三つ目の行列はモデル化がうまくいかないと思われる行列だが、全体での正答率は 80%前後とそれほど悪くなく、一つ目の行列とあまり変わらない。ただし、本来失敗するパターンを誤って成功に分類する場合が多くなってしまっており、また他の b でもどちらかの正答率が極端に小さくなってしまうという結果だった。

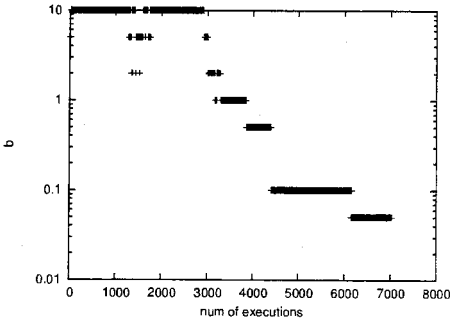
5. まとめと課題

ソフトウェアの自動チューニングにおいて、実行結果を成功失敗に分類することによりパターン認識を応用して実験数を減少させる手法を提案した。識別モデルには単純ベイズを用いて、Lis による線形疎行列求解の収束性を例題として実験を行ったが、多くの行列について失敗するパターンの 7 割程度を削減できた。

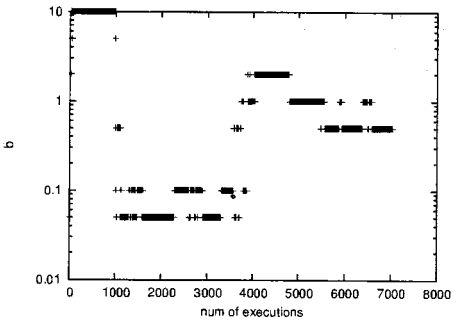
実験パターンの選択順序そのものは恐らく本質的ではないと思われるが、パターンの偏りによっては終了条件の設定などがより難しくなるという問題があるようだ。

また成功数の割合や実験数にはトレードオフが存在し、それを評価する指標がないこと、またそれを制御するのが難しいことが未解決である。

モデルの評価については、全体での正答率だけでは把握できず、成功失敗それぞれの正答率も重要な意味

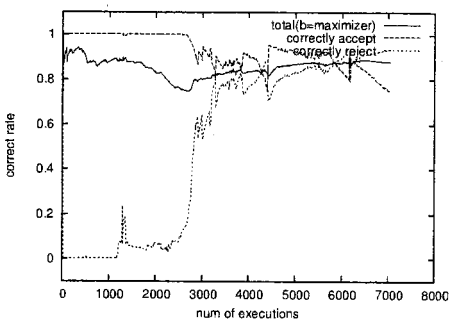


ACUSIM/Pres_Poisson

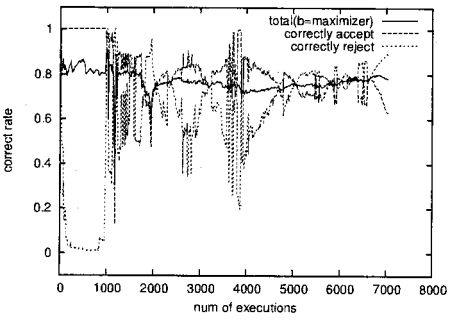


HB/bcsstk14

図 4　高確率パターンを優先して実行する場合の経過



ACUSIM/Pres_Poisson



HB/bcsstk14

図 5　高確率パターンを優先して実行する場合の経過

をもっていることがわかった。

その他の今後の課題としては、他のモデル、例えばSVM や MEM などの利用が挙げられるが、解法の名称などそれ自体にはあまり意味のない非数値データの扱いをどのようにするかは問題である。また、特徴量選択/抽出についても今後必要となるだろう。

謝　辞

本研究は、文部科学省科学研究費補助金、特定領域研究「情報爆発」の補助の下で実施しました。また、実験の一部は同提供のプラットフォーム InTrigger をお借りして行いました。

参 考 文 献

1) Tanaka, T., Katagiri, T. and Yuba, T.: d-Spline Based Incremental Parameter Estimation in Automatic Performance Tuning, *Proceedings of PARA'06, CP4* (2006).
2) 小谷和正, 須田礼仁：汎用的なソフトウェア自動チューニング機構のための実験計画法の応用の検討, 情報処理学会 研究報告, 2006-HPC-107, pp. 193–198 (2006).
3) 小武守恒, 藤井昭宏, 長谷川秀彦, 西田　晃：Lis: a Library of Iterative Solvers for linear systems, High-Performance Computing Symposium (ポスター), p.30 (2006).
4) : SSI Project, Lis home page, `http://ssi.is.s.u-tokyo.ac.jp/lis/`.
5) A.K.Jain, R.P.W.Duin and J.Mao: Statistical Pattern Recognition: A Review, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp.4–37 (2000).
6) Bishop, C.M.: *Pattern Recognition and Machine Learning*, Springer-Verlag (2006).
7) : The University of Florida Sparse Matrix Collection, `http://www.cise.ufl.edu/research/sparse/matrices/`.