ORSA Journal on Computing
Vol. 4, No. 3, Summer 1992

0899-1499/92/0403-0311 $01 25

Simulated Annealing with a Temperature Dependent Penalty Function

JULIO M. STERN / *School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853, jstern@cs.cornell.edu and the Computer Science Department of the University of São Paulo, Brazil, jstern@ime.usp.br*

(Received: June 1990; final revision received: December 1990; accepted: November 1991)

We formulate the problem of permuting a matrix to block angular form as the combinatorial minimization of an objective function. We motivate the use of simulated annealing (SA) as an optimization tool. We then introduce a heuristic temperature dependent penalty function in the simulated annealing cost function, to be used instead of the real objective function being minimized. Finally we show that this temperature dependent penalty function version of simulated annealing consistently outperforms the standard simulated annealing approach, producing, with smaller running times, better solutions. We believe that the use of a temperature dependent penalty function may be useful in developing SA algorithms for other combinatorial problems.

Reducing a matrix to block angular form (BAF) is important for a variety of decomposition methods. In Section 1 we formulate the BAF reduction problem as a combinatorial partition problem, where the cost of a given partition or state, $cost(p)$, is the objective function to be minimized.

In Section 2 we present a "standard" simulated annealing (SSA) algorithm to approximately solve this combinatorial problem. The SSA can be seen as a generalization of simulated annealing (SA) algorithms for the graph partitioning problem.[14]

In Section 3 we motivate the use of a heuristic cost function,

$$cost(p, \mu(t)) \equiv cost(p) - \mu(t)\, penalty(p),$$

in the SA, instead of the real objective function being minimized, $cost(p)$. Parameter μ is temperature dependent, and only in the zero temperature limit we have $\mu = 0$ and $cost(p, 0) = cost(p)$. The main reasons for using $cost(p, \mu)$ are:

- it is only marginally more expensive to compute than $cost(p)$;
- it helps us "sense" the proximity of low cost states;
- it breaks down massively degenerate states of $cost(p)$ that can "trap" the SSA in nonoptimal metastable states.

In Sections 4 and 5 we report numerical experiments. In these experiments the temperature dependent penalty function simulated annealing (TPSA) consistently outperforms the SSA, producing much better solutions with smaller running times. In Section 5 we also relate the temperature dependent penalty function, that can be seen as perturbing the metric of the original problem, to other variants of the SA algorithm. Finally in Section 6 we compare SA to another heuristic to solve the BAF reduction problem.

1. The Block Angular Form

The Column Block Angular Form (CBAF) reduction problem is, by rows and columns permutations, P and Q, to reduce a given matrix A, $m \times n$, to CBAF; i.e., we want PAQ with b diagonal rectangular blocks, $B_1, \ldots, B_b$, plus some residual columns C (Figure 1). This can also be seen as a hypergraph partition problem where we paint all nonzero elements (NZEs) of each vertex or row, $i \in M \equiv \{1, \ldots, m\}$, with a color $p(i) \in B \equiv \{1, \ldots, b\}$. The color $q(j)$ of an edge or column $j \in N \equiv \{1, \ldots, n\}$, is then the set of all its NZE's colors, and multicolored edges of the hypergraph correspond to residual columns in the CBAF.

A more General Block Angular Form (GBAF), also allows some residual rows, B_0, to remain uncolored, or to receive the color 0 (Figure 2). Finally the Row Block Angular Form (RBAF) only allows residual rows (Figure 3). Our interest in reducing a matrix to a BAF relates to techniques for sparse matrix computations,[7, 21, 25, 26, 30] and some other general decomposition methods. In these applications we always want:

1. Roughly the same number of rows in each block.
2. Only a few residual rows or columns.

From 1 and 2 it is natural to consider the minimization of the function

$$cost(p) \equiv \alpha \sum_{k=1}^{b} [m/b - s(k)]^2 + \beta c(p) + \gamma r(p)$$

$$c(p) \equiv |\{j \in N: |q(j)| \geq 2\}|$$

$$r(p) \equiv |\{i \in M: p(i) = 0\}|$$

$$s(k) \equiv |\{i \in M: p(i) = k\}|.$$

The first term in $cost(p)$ measures the deviation of each block from the ideal size m/b; $c(p)$ is the number of residual columns, and $r(p)$ is the number of residual rows.

In most applications one can view the diagonal blocks as

Subject classifications: Programming: integer; heuristics; mathematics: matrices; networks/graphs. heuristics.

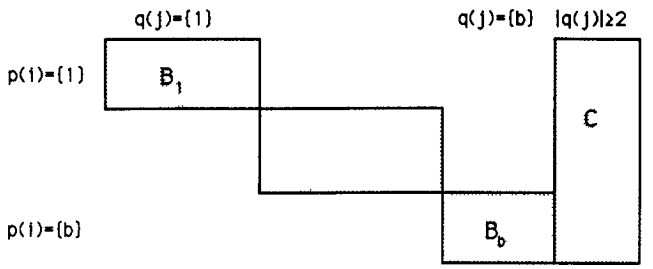


Figure 1. CBAF.

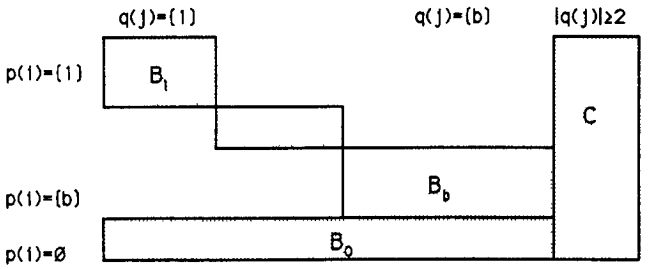


Figure 2. GBAF.

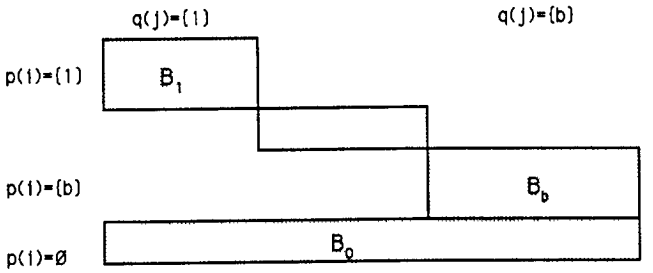


Figure 3. RBAF.

subproblems that can be solved independently, and the residual rows and columns as a final step, or masterproblem, that can not start until the subproblems are complete. Usually the BAF reflects a natural substructuring of the original problem: subsystems in a power or data network, intra and interdepartmental transactions in a company, dependence on sectorial and global econometric indices, substructures of a truss, etc.

The next sections address the block angular form reduction problem for a general rectangular matrix. For clarity, we will restrict ourselves to CBAF. Even very special cases of the CBAF reduction problems are NP-hard. For example: take A, the incidence matrix of a graph, m an exact multiple of $b = 2$, normalize $\beta = 1$, and take α big enough to force all blocks to have exactly m/b rows. CBAF then becomes the exact-graph-2-partitioning problem, and the recognition version of this problem is known to be NP-complete; see [10] (problem ND14 on page 209).

2. Standard Simulated Annealing

The NP-hardness of the CBAF reduction problem, and the many degrees of freedom in its formulation, suggests Simulated Annealing (SA) as an optimization tool. We follow the approach of Johnson et al.[14] From the matrix A, it suffices to have, for any row i, a list of columns in which there are NZE's, i.e.

$$aijs(i) \equiv \{j \in N: A[i,j] \neq 0\}$$

We begin the standard simulated annealing (SSA) setting an initial random state or coloring $p: M \to B$. We then compute the number of NZEs of each color per column, i.e., the $n \times b$ matrix of weights:

$$W[j,k] \equiv |\{i \in M: A[i,j] \neq 0 \wedge p(i) = k\}|$$

and from W it is trivial to compute $s(k)$ and $cost(p)$.

From state p we *propose* a possible *move*. In our case a move is a random change in the color of a row

$$p(i) \mapsto p'(i) \in B$$

The set of states p' reachable from p by a single move is the *neighborhood*, $nbd(p)$ of p. In our problem, any state p has a neighborhood of the same size, *NBDSIZE*. We can choose one of the m rows, and then choose one of the b colors for this row, so:

$$NBDSIZE = |nbd(p)| = m * b$$

Next compute the cost of this proposed move Δ_0.

$$\Delta_0 \equiv cost(p') - cost(p)$$

The move is then accepted with probability $prob(\Delta_0)$

$$prob(\Delta_0) \equiv \begin{cases} 1 & \text{if } \Delta_0 \leqslant 0 \\ \exp(-\Delta_0/temp) & \text{if } \Delta_0 \geqslant 0 \end{cases}$$

To compute Δ_0 it suffices to scan $W[j,k]$ for $j \in aijs(i)$. If the proposed move was accepted we update p, W, *and* $cost(p)$. The temperature parameter *temp* is initially set at $temp = STARTTEMP$.

New moves are proposed until we either:

- make $LENGTH \equiv SIZEFACTOR * NBDSIZE$ proposals, or
- accept $CUTOFF * LENGTH$ moves.

Then we go to a *cooling* step, where we:

- compute the *acceptance rate* for this temperature:

$$accrate \equiv |accepted\ moves|/|proposals|$$

- cool the temperature setting $temp \leftarrow temp * TEMPFACTOR$

The SSA is said to be *frozen* at a given temperature if $accrate < MINACCEPT$. The algorithm terminates after *FROZENMAX* consecutive frozen temperatures.

The parameters *STARTTEMP*, *TEMPFACTOR*, *SIZEFACTOR*, *CUT-OFF*, *MINACCEPT*, and *FROZENMAX* are set by the user, as discussed in [14] and Section 4 of this

paper. Our reliance on [14] to define a "standard" SA is convenient for our study, but by no means the only possibility. Alternative approaches, methodologies and views of the field can be found in [1, 2, 5, 6, 20 and 29].

3. A Temperature Dependent Cost Function

The SSA behaves poorly in the CBAF problem, mainly because it is very difficult to sense the proximity of low cost states, i.e.:

1. Most of the neighbors of a low cost state, p, can have much higher costs.
2. The problem is massively degenerate, i.e., often a "connected by single moves" set of equal cost states, $S = \{p_1, \ldots, p_d\}$, has such a large degeneracy d, that, even rejecting all proposals that would take us out of S, would still give us a significant acceptance rate.

Difficulty 2 implies, in particular, the failure of the SSA termination criterion: A degenerate locally minimum connected component of $S_c \equiv \{p, cost(p) = c\}$, could trap the SSA into forever sustaining an acceptance rate above the threshold *MINACCEPT*.

The best way we found to overcome difficulties 1 and 2 is to use a temperature-dependent cost function:

$$cost(p, \mu(t)) \equiv cost(p) - \mu(t)\, penalty(p)$$
$$penalty(p) \equiv \sum_{j, |q(j)|>1} (b - |q(j)|)$$

The additional term in $cost(p, \mu)$ can be seen as an heuristic penalty function that rewards multicolored columns for using fewer colors. This penalty function, and some possible variants, are inspired by the *tally* function $\theta(j)$ used in the *P3* heuristic[12] for sparse *LU* factorization. The temperature dependent parameter $\mu(t)$ gives the relative weight of the penalty function in $cost(p, \mu)$.

Function $cost(p, \mu)$ also has the following properties:

1. $cost(p, 0) = cost(p)$
2. $cost(p, \mu)$ is linear in μ.

Properties 1 and 2 suggest that we can cool the parameter μ along with the temperature, much in the same way we decrease a parameter of the barrier functions in some constrained optimization algorithms.[19] We use $cost(p, \mu)$ in the temperature-dependent penalty function simulated annealing (TPSA)[24] as follows:

- Initially set $\mu = STARTMU$;
  set p an initial partition, and initialize w, s, c, and $cost = cost(p)$
- For each proposal compute

$$\Delta_0 \equiv cost(p') - cost(p)$$
$$\Delta_\mu \equiv cost(p', \mu) - cost(p, \mu)$$

- Accept the move with probability $prob(\Delta_\mu)$
  and then update w, s, c, and $cost = cost + \Delta_0$
- Keep track of the best, i.e. smallest $cost(p)$, solution found
- At each cooling step set $\mu \leftarrow \mu * MUFACTOR$.

To compute Δ_μ and Δ_0 we scan the same columns of W that would be scanned to compute Δ_0 alone. It only takes a few more arithmetic operations to compute Δ_μ along with Δ_0, which has little impact on the running time of the SA. The $cost(p, \mu)$ is never computed (only the differentials Δ_μ).

The main goals for the temperature dependent penalty function, namely to accelerate the SA convergence to the global optimum and to avoid premature convergence to locally optimal solutions, have motivated many others interesting strategies. Closest to our approach, directly involving the cost function, are [23] and [28]. Alternative approaches deal with the cooling schedules, [13] and [31], or with the topology of the neighborhood, [11] and [17]. We believe that the TPSA method, which can be seen as a perturbation on the metric of the problem, could be useful in developing SA algorithms for other problems where the user is aware of some heuristic merit or penalty function. Also, there is no a priori impediment to combining metric perturbations with better topologies (i.e., neighborhood structures) or improved cooling schedules.[27]

4. Numerical Experiments

We tested the SSA and the TPSA for CBAF reduction on three different matrices. These matrices are coefficient matrices from linear programs in the NETLIB collection of test problems. A portrait of the sparsity structure of these matrices can be found in [18].

NETLIB LP	m	n	NZEs	Matrix Structure
scatp1	300	660	1872	9 steps stair case
scfxm1	330	600	2732	4 diagonal blocks of diff. sizes
grow15	300	645	5620	narrow band with dense clusters

For each matrix we used all possible combinations of the parameters:

$STARTMU \in \{0.5, 1.0\}$ and $MUFACTOR \in \{0.90, 0.95, 0.98\}$ plus $STARTMU = 0.0$ that corresponds to the SSA, in which case *MUFACTOR* is irrelevant. The cost function parameters were chosen according to our specific applications,[25, 26] always $b = 4$, $\alpha = 0.01$ and $\beta = 1$. Finally we always had $SIZEFACTOR = 16$, $CUTOFF = 0.125$, $TEMPFACTOR = 0.95$, and *STARTTEMP* was set to give us an initial acceptance rate of $\approx 40\%$. We set these last four parameters following the guidelines in [14], but the relative performances of the various annealings do not depend heavily on them.

To avoid the problems with termination criteria mentioned in Section 2, we gave each SA run a fixed "budget" of $1.0E6$ (1.0×10^6) proposals. In each run, after completion of $0.1E6$, $0.2E6$, $0.3E6$, $0.4E6$, $0.5E6$ and $1.0E6$ proposals, we recorded *bestcost* = the best $cost(p)$ already found, and *totmove* = the total (cumulative) number of accepted

moves. With *bestcost* we can monitor the progress of the annealing; *totmove* and *totprop* (the total number of proposals) are measures of the running time of the SA. Variable *totprop* is more important for a pure sequential implementation, while *totmove* may dominate in parallel machines.

In each SA run, the initial state, row selection, color change and acceptance processes were controlled by 4 different streams of a C-coded 100-streams 32-bits linear congruential random number generator. For each matrix and parameter setting we performed an experiment consisting of 82 runs, each controlled by a different 4-stream set.

5. Experimental Results

Our results are summarized by the box-plots of *bestcost* and *totmove* for each experiment (Figures 4–9). The three horizontal lines of a box, inside, lower and upper ends, correspond to the median, lower and upper quartiles; from the ends of the box "whiskers" may extend for up to 1.5 times the inter-quartile distance, and beyond that, detached points indicate extreme outliers.[3] Each figure contains seven sequences of five plots. Each sequence corresponds to budgets of $0.1E6$, $0.2E6$, $0.3E6$, $0.5E6$ and $1.0E6$ proposals. The seven sequences correspond to the parameters *STARTMU* and *MUFACTOR* set to:

$$[0.0, *]\ [0.5, 0.90]\ [0.5, 0.95]\ [0.5, 0.98]$$
$$[1.0, 0.90]\ [1.0, 0.95]\ [1.0, 0.98]$$

As noted in the last section, the leftmost sequence of plots correspond to the SSA, since $STARTMU = 0.0$.

From the box-plots we see that the TPSA outperforms the SSA for all of our test matrices, for all TPSA parameter setting, and all budgets we tried. Also, the TPSA usually continued to make progress with bigger budgets, while the SSA "saturated" much earlier. Moreover, for a given *totprop* budget, the *totmove* component of the running time is usually smaller in the TPSA than in the SSA!

For the SSA in matrix grow15, note the convergence of *bestcost* to a very precise level $\blacktriangleright$ $min\{cost(p)\}$. Most of the connected components of the macro-state $S_c \equiv \{p, cost(p) = c\}$ are not only massively degenerate, but also local minima, which give us an intuitive explanation of the metastable character of S_c.[4] The penalty term in $cost(p, \mu)$ decreases degeneracy and local minima overall, thereby destroying the metastability of S_c.

The ratio *MUFACTOR/TEMPFACTOR* is very important for the TPSA performance. In our experiments we set *MUFACTOR* to 0.90, 0.95, and 0.98, respectively a slow, neutral, or fast cooling of μ relative to *TEMPFACTOR* = 0.95. The slow cooling of μ very effectively avoids degeneracy, at the expense of optimizing a biased $cost(p, \mu)$ with a relatively big penalty factor μ. In Table I we show some statistics of the TPSA performance. Each line of Table I corresponds to a series of 82 independent annealings, for a given test matrix and setting of parameters *STARTMU* and *MUFACTOR*. The statistics in Table I are: *aoc*, the approxi-

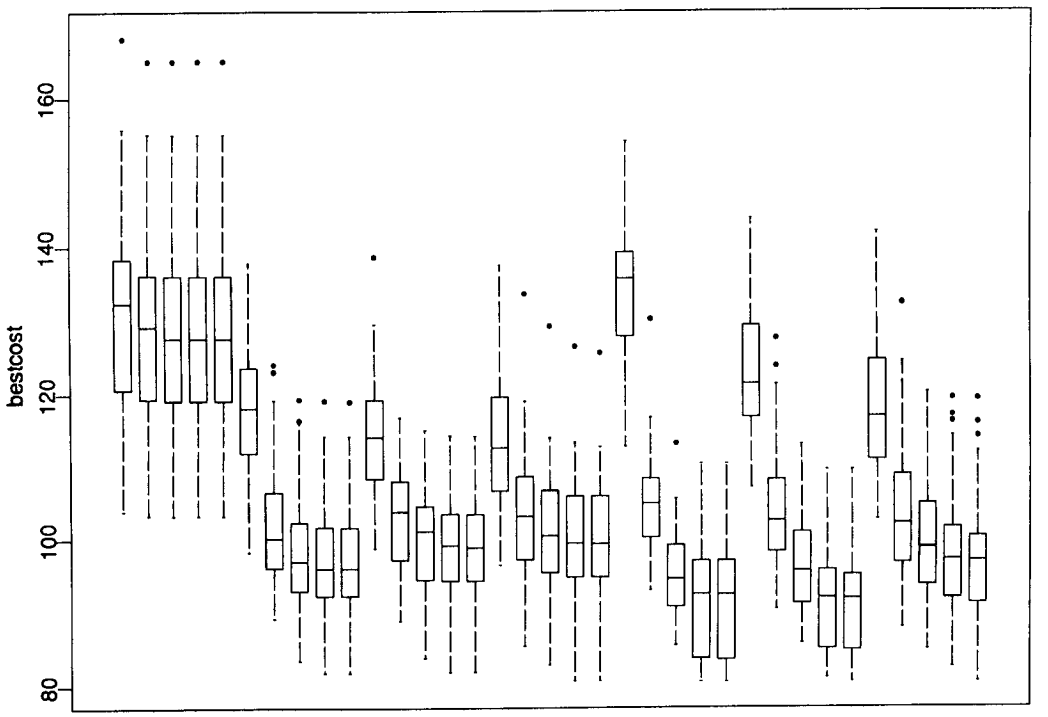
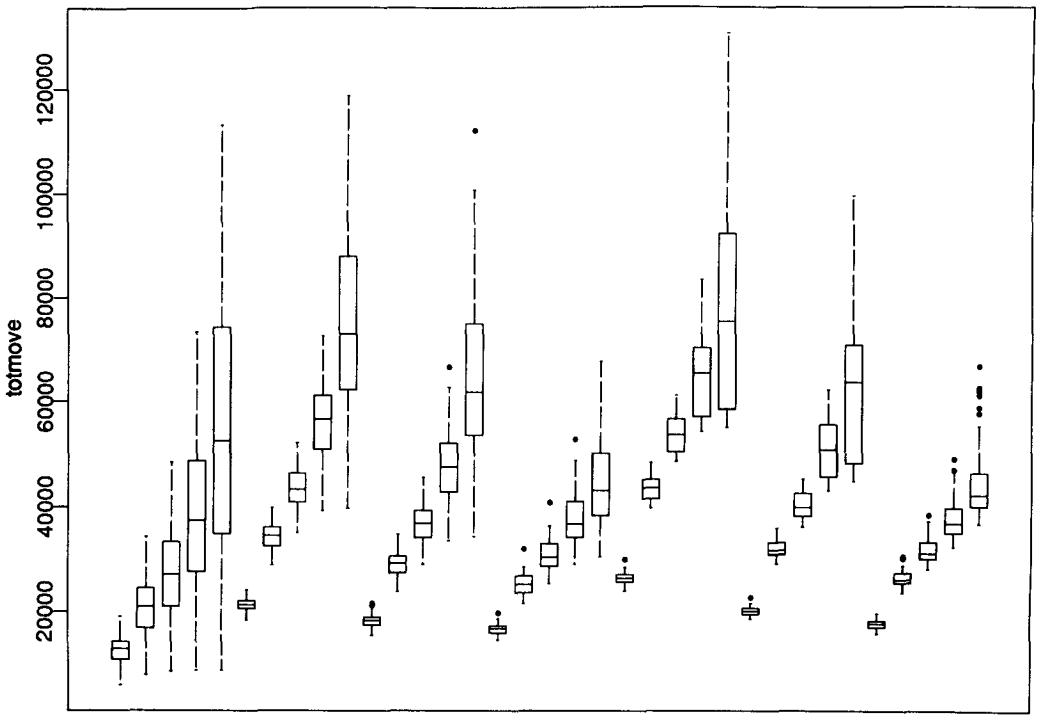
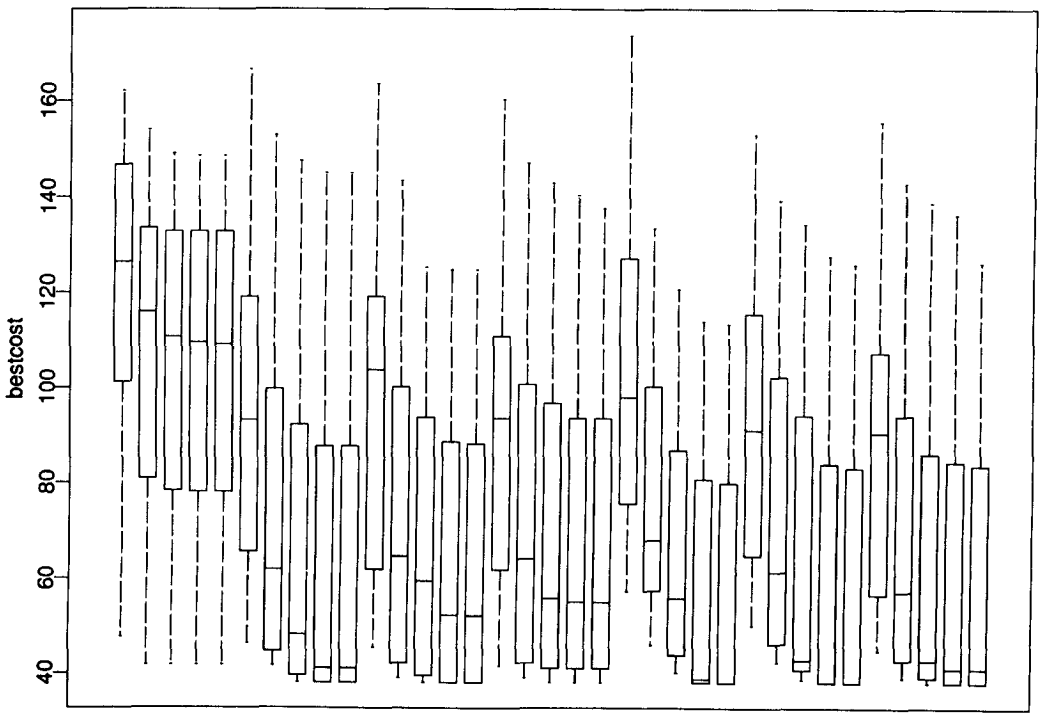


Figure 4. *bestcost* for matrix sctap1.



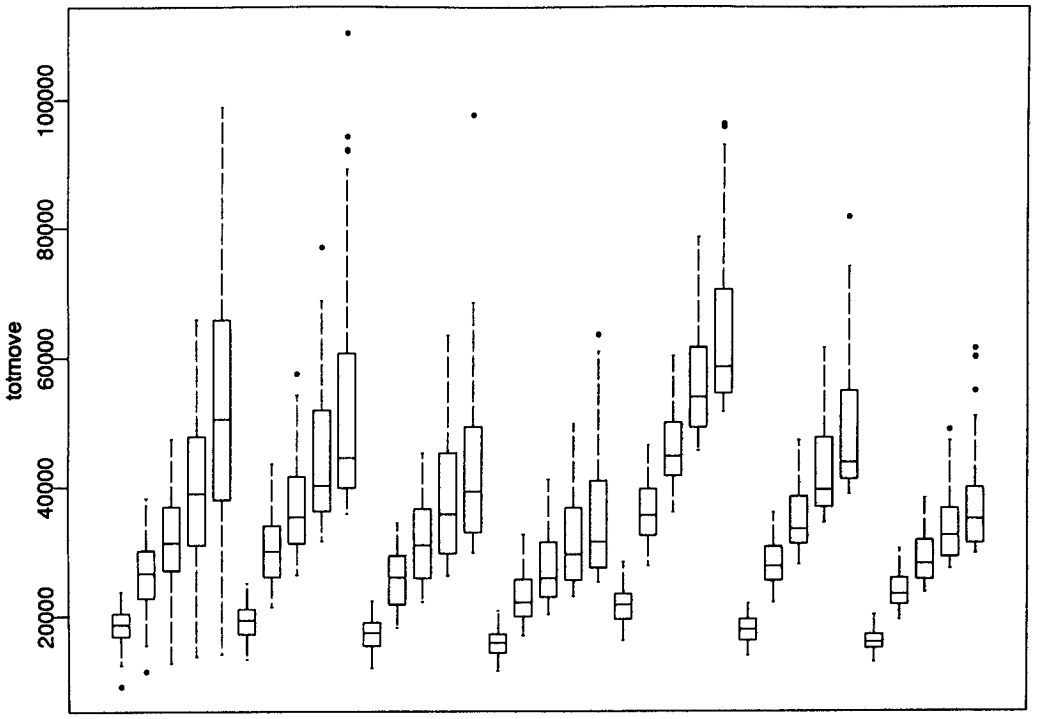
SARTMU={0.0,0.5,1.0} X MUFACTOR={0.90,0.95,0.98} X budget={1,2,3,5,10}E5

Figure 5. *totmove* for matrix sctap1.



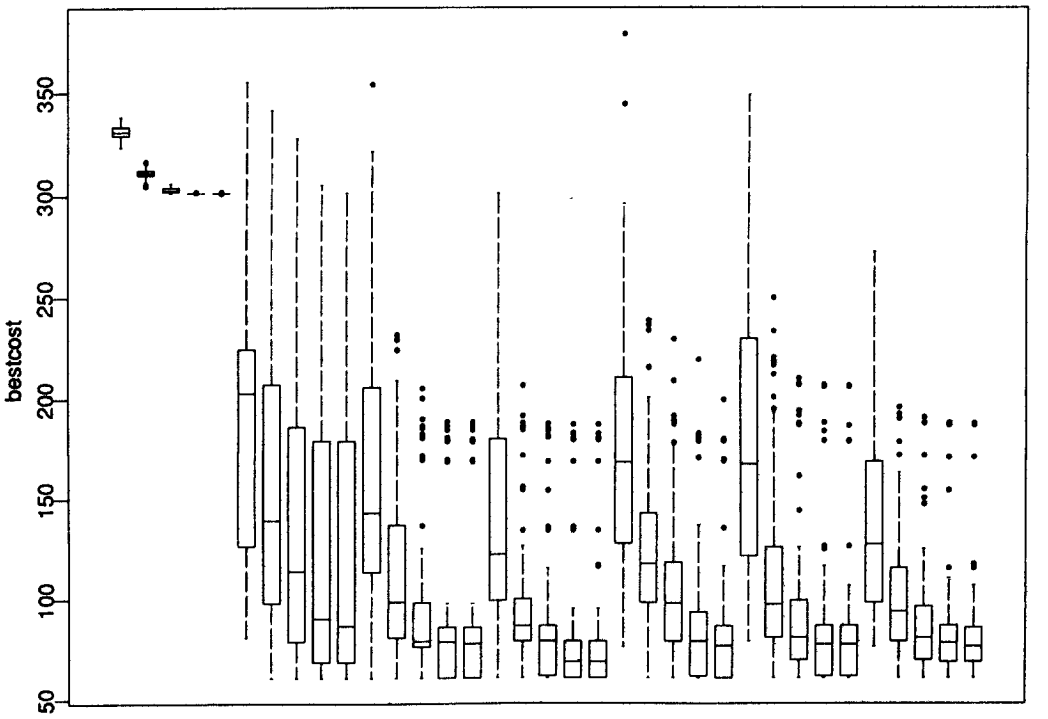
STARTMU={0.0,0.5,1.0} X MUFACTOR={0.90,0.95,0.98} X budget={1,2,3,5,10}E5

Figure 6. *bestcost* for matrix scfxm1.



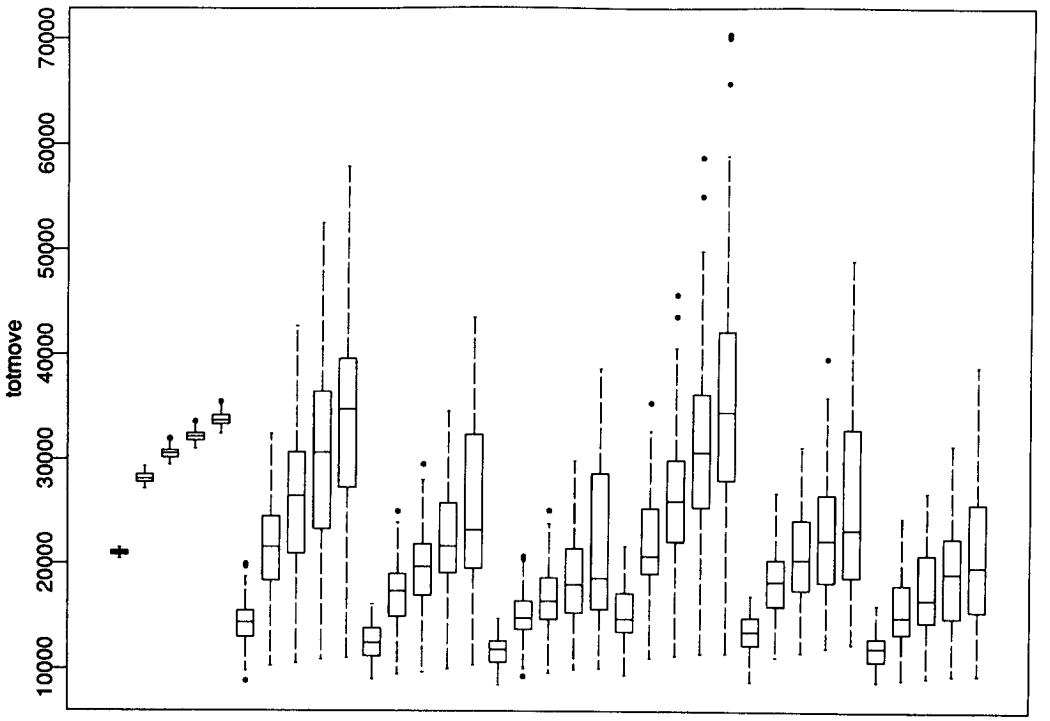
SARTMU={0.0,0.5,1.0} X MUFACTOR={0.90,0.95,0.98} X budget={1,2,3,5,10}E5

Figure 7. *totmove* for matrix scfxm1.



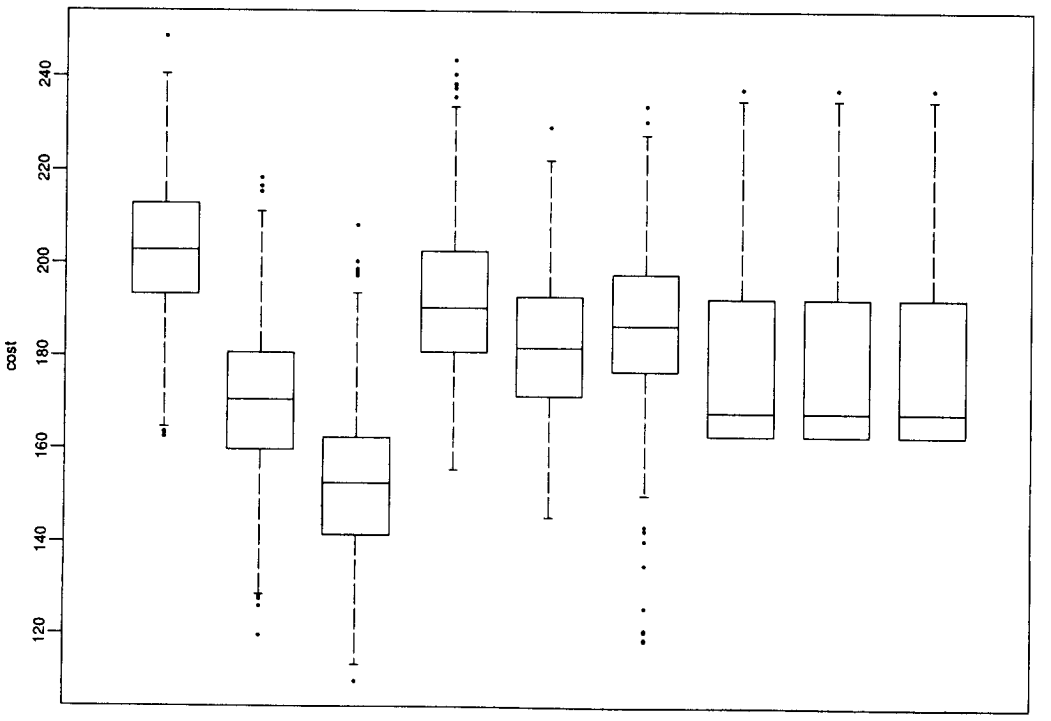
STARTMU={0.0,0.5,1.0} X MUFACTOR={0.90,0.95,0.98} X budget={1,2,3,5,10}E5

Figure 8. *bestcost* for matrix grow15.



SARTMU={0.0,0.5,1.0} X MUFACTOR={0.90,0.95,0.98} X budget={1,2,3,5,10}E5

Figure 9. *totmove* for matrix grow15.



matrices{sctap1, scfxm1, grow15} X MU={0 0, 0 5, 1 0}

Figure 10. Contour heuristic costs.

Table I. TPSA Dependence on STARTMU and MUFACTOR

SMU	*MUF*	*aoc*	*naoc*	*mbc*	*sdbc*	mtm
		Matrix sctap1				
0.0	*	103.3	1	128.1	12.3	54972
0.5	0.90	81.8	2	96.6	7.7	74578
0.5	0.95	81.9	2	98.7	6.7	64098
0.5	0.98	80.6	1	99.4	8.2	45460
1.0	0.90	80.5	1	91.2	7.1	79760
1.0	0.95	80.5	4	91.2	7.2	61654
1.0	0.98	80.5	4	96.1	8.5	44843
		Matrix scfxm1				
0.0	*	41.8	1	103.6	30.9	51240
0.5	0.90	38.1	40	62.6	30.3	52136
0.5	0.95	38.1	25	65.3	27.8	42667
0.5	0.98	38.1	20	68.3	29.6	34313
1.0	0.90	38.1	44	55.3	23.2	63759
1.0	0.95	38.1	47	57.2	26.7	48695
1.0	0.98	38.1	38	59.2	27.0	36900
		Matrix grow15				
0.0	*	300.0	82	300.0	0.0	29642
0.5	0.90	63.0	24	114.5	57.6	30359
0.5	0.95	63.0	27	92.0	40.2	23956
0.5	0.98	63.0	40	82.1	39.0	19733
1.0	0.90	63.0	24	98.6	44.3	37525
1.0	0.95	63.0	29	82.1	29.4	23668
1.0	0.98	63.0	21	85.7	32.8	20941

mate optimal cost, i.e., the smallest *bestcost* found in the 82 annealings; *naoc*, how many times the approximate optimal cost was found; *mbc* and *sdbc*, the mean and standard deviation of *bestcost*; and *mtm*, the mean of *totmove*.

6. Alternative Heuristics

There are in the literature several alternative heuristics to special cases of our original problem, i.e., the CBAF reduction or hypergraph partition problem.

The most common restriction of the original problem are:

1. *Graph-Partition*: A is the incidence matrix of a graph, i.e. A has only two nonzero elements (NZEs) per column.
2. *2-Partition*: There are only two blocks, i.e. $b = 2$.
3. *Exact-Partition*: Each block must have exactly m/b rows, i.e., $\beta = \infty$.

Examples of such heuristics, with the restrictions applied to the original problem listed in parentheses, are: Kernighan and Lin[16] (1, 2, 3); Fiduccia and Mattheyses[9] (1, 2); Sangiov-Vincentelli and Chua[22] (1); and Exposito and Franquelo[8] (1). We present a generalization of the "Contour" heuristic (CH), as described in [22]. Exposito and Franquelo[8] give more efficient variations of the CH for graphs and highlight some of the connections between the graph partition problem and computational linear algebra.

The *cut* of a block, $cut(B)$, is defined as the set of hyper-edges incident to vertices in the block *and* to vertices in its complement (or the set of columns of A with NZE's in rows of B *and* in rows outside B). The cost of a block is defined as the cardinality of its cut, i.e.

$$cost(B) \equiv |cut(B)|.$$

The CH forms the row blocks in A (vertex clusters in the hypergraph) one at a time. To form the first block, B, we take at random the first vertex. At step $t, t = 2, 3 \ldots$ we then add to the block a vertex v_t that minimizes $cost(B_t) = cost(B_{t-1} + v_t)$. For each t we store v_t and $cost(B_t)$. Our final step is to select at which $\hat{t}$ to "terminate" the block $B = B_{\hat{t}}$. To avoid having too small or too large a block, we impose

$$(1 - \omega)m/b \leqslant \hat{t} \leqslant (1 + \omega)m/b$$

where the parameter $\omega \in [0, 1]$ is set by the user, and then take $\hat{t}$ so to minimize $cost(B_{\hat{t}})$. After we finish block B we eliminate from A the rows of B, and the columns having any NZEs in rows of B. If $b > 1$ we use the CH to form a new block in the reduced A, with parameters $m \leftarrow m - \hat{t}$ and $b \leftarrow b - 1$.

As in the TPSA, we tried to improve the performance of the CH by using a heuristic penalty function. When selecting the vertices for the sequence $v_1, v_2, \ldots, v_{\lfloor (1+\omega)m/b \rfloor}$, we use the perturbed cost function:

$$cost(B_t, \mu) \equiv cost(B_t) - \mu * penalty(B_t)$$

$$penalty(B_t) \equiv |\{e \in cut(B_t),$$

$$e \text{ adjacent to exactly 1 vertex not in } B\}|.$$

The interpretation of $penalty(B_t)$ is similar to the penalty term in the TPSA, and even closer to the *tally* function of Hellerman and Rarick.[12]

We tested the CH on our three test matrices. We tested parameter $\mu \in \{0.0, 0.5, 1.0\}$. The case $\mu = 0.0$ corresponds to the unperturbed block cost. Experimentation indicated that taking $\omega = 0.2$ was a good choice. We used the CH to form 3 blocks, considering the remaining vertices as the fourth block, and then evaluated the final configuration with the cost function $cost(p)$, as described in Section 1, and using the parameters given in Section 4. In Figure 10 we have box-plots of these final costs. For each test matrix and parameter setting, we ran the CH 600 times, which demands a computing time roughly equivalent to 10 runs of the SA. Although the CH outperforms the SSA for one of the test matrices (grow15), the TPSA clearly outperforms the CH in all cases. Also the "response" of the CH performance to the value of parameter μ is not as uniform as in the TPSA.

Acknowledgments

This work was supported by grant 87 0540-9 from FAPESP, Fundação de Amparo à Pesquisa do Estado de São Paulo, and the Computer Science Department of the University of São Paulo, Brasil. I am grateful to Stephen A. Vavasis and Russell R. Barton, from Cornell University, for many useful suggestions.

References

1. E. Arts and J. Korst, 1989. *Simulated Annealing and Boltzmann Machines*, John Wiley, Chichester.

2. D.H. ACKLEY, 1987. *A Connectionist Machine for Genetic Hillclimbing*, Kluwer, Boston.
3. R.A. BECKER and J.M. CHAMBERS, 1984. *S An Interactive Environment for Data Analysis and Graphs*, Wadsworth Statistics/Probability series, Pacific Grove.
4. K. BINDER, 1986. *Monte Carlo Methods in Statistical Physics* (Topics in Current Physics 7), Springer-Verlag, Berlin.
5. K. BINDER and D.W. HEERMANN, 1988. Monte Carlo Simulation in Statistical Physics (Series in Solid-State Science 80), Springer-Verlag, Berlin.
6. N.E. COLLINS, R.W. EGLESE and B.L. GOLDEN, 1988. *Simulated Annealing, An Annotated Bibliography*, in Johnson.[15]
7. I.S. DUFF, 1986. *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford.
8. A.G. EXPÓSITO and L.G. FRANQUELO, 1987. A New Contribution to the Cluster Problem, *IEEE Transactions on Circuits and Systems 34*, 546–552.
9. C.M. FIDUCCIA and R.M. MATTHEYSES, 1982. A Linear Time Heuristic for Improving Network Partitions, *IEEE Design Automation Conferences 19*, 175–181.
10. M.R. GAREY and D.S. JOHNSON, 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, New York.
11. L. GOLDSTEIN and M. WATERMAN, 1988. *Neighborhood Size in the Simulated Annealing Algorithm*, in Johnson.[15]
12. E. HELLERMAN and D.C. RARICK, 1971. Reinversion with the Preassigned Pivot Procedure, *Mathematical Programming 1*, 195–216.
13. M.D. HUANG, F. ROAMEO and A. SANGIOVANNI-VINCENTELLI, 1986. An Efficient General Cooling Schedule for Simulated Annealing, Proceedings IEEE Internation Conference on Computer-Aided Design, pp. 381–384.
14. D.S. JOHNSON, C.R. ARAGON, L.A. MCGEOCH and C. SCHEVON, 1989. Optimization by Simulated Annealing: An Experimental Evaluation, Part 1, *Operations Research 37*, 865–892.
15. M.E. JOHNSON (Editor), 1988. *Simulated Annealing & Optimization*. American Science Press, Syracuse, NY (This book is also the volume 8 of the American Journal of Mathematical and Management Sciences.)
16. B.W. KERNIGHAN and S. LIN, 1970. An Efficient Heuristic Procedure for Partitioning Graphs, *The Bell System Technical Journal 49*, 291–307.
17. M. LUNDY and A. MEES, 1986. Convergence of an Annealing Algorithm, *Mathematical Programming 34*, 111–124.
18. I.J. LUSTIG, 1987. An Analysis of an Available set of Linear Programming Test Problems, Technical Report SOL-87-11, Department of Operations Research, Stanford University.
19. G.P. MCCORMICK, 1983. *Nonlinear Programming: Theory, Algorithms and Applications*, John Wiley, Chichester.
20. R.H.J.M. OTTEN and L.P.P.P. VAN GINNEKEN, 1989. *The Annealing Algorithm*, Kluwer, Boston.
21. R.J. PLEMMONS and R.E. WHITE, 1990. Substructuring Methods for Computing the Nullspace of Equilibrium Matrices, *SIAM Journal on Matrix Analysis and Applications 11*, 1–22.
22. A. SANGIOVANNI-VINCENTELLI and L.O. CHUA, 1977. An Efficient Heuristic Cluster Algorithm for Tearing Large-Scale Networks, *IEEE Transactions on Circuits and Systems 24*, 709–717.
23. C. SECHEN and K. LEE, 1987. An Improved Simulated Annealing Algorithm for Row-Based Placement, Proceedings IEEE International Conference on Computer-Aided Design, pp. 478–481.
24. J.M. STERN, 1990. *Simulated Annealing with a Temperature Dependent Cost Function*, Technical Report CCOP-90-1, Cornell Computational Optimization Project, Cornell University, Ithaca.
25. J.M. STERN and S.A. VAVASIS, 1990. Nested Dissection for Sparse Nullspace Bases, Technical Report TR-90-1173, Department of Computer Science, Cornell University, Ithaca.
26. J.M. STERN and S.A. VAVASIS, 1991. Active Set Algorithms for Problems in Block Angular Form, in preparation for 1991.
27. J.M. STERN, 1992. *Adaptive Cooling for Simulated Annealing with Temperature Dependent Penalty Functions*, in preparation for 1992.
28. C.A. TOVEY, 1988. *Simulated Simulated Annealing*, in Johnson.[15]
29. P.J.M. VAN LAARHOVEN and E.H.L. AARTS, 1987. *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht.
30. R.L. WEIL and P.C. KETTLER, 1971. Rearranging Matrices to Block Angular Form for Decomposition Algorithms, *Management Science, 18:1*, 98–108.
31. S.R. WHITE, 1984. Concepts of Scale in Simulated Annealing, *American Institute of Physics Conference Proceedings 122*, 261–270.