International Business Machines Corp. (1991). *Optimization Subroutine Library: Guide and Reference.*

I.J. Lustig, R.E. Marsten, D.F. Shanno (1989). *Computational Experience with a Primal Dual Interior Point Method for Linear Programming.* Technical Report SOR 89-17, Princeton University, Princeton, NJ.

I.J. Lustig, R.E. Marsten, D.F. Shanno (1990). *On implementing Mehrotra's Predictor-Corrector Interior Point Method for Linear Programming.* Technical Report SOR 90-03, Princeton University, Princeton, NJ.

B. Stroustrup (1986). *The C++ programming language.* Addison Wesley, Reading, Massachusetts.

J.A. Tomlin, J.S. Welch Jr. (1993). Mathematical Programming Systems. E. Coffman, J.K. Lenstra (ed.). *Handbook of Operations Research and Management Science: Computation*, North-Holland, Amsterdam, forthcoming.

J.S. Welch Jr. (1987). The data management needs of mathematical programming applications. *IMA J. of Mathematics in Management 1*, 237-250.

An infeasible dual affine scaling method for linear programming

Knud D. Andersen
Dept. of Math. and Computer Sci.
Odense University
Campusvej 55, 5230 Odense M
Denmark

July 15, 1993

Abstract

In this paper we describe an infeasible dual affine scaling method. The method can be view as a dynamic "big-M" method but it actually never calculate the "big-M". We show that in contrast to the "big-M" method this method always finds a feasible point. The method is shown to work well on the standard test set "netlib", and on a special test problem with many free variables arising in plastic, collapse analysis. In particular, the method provides improved results for LP-problems without interior points in the dual space, on which in theory the dual algorithm should not work. We also present an implementation of an interior point algorithm. Numerical results are presented, running at the CONVEX vector computer C3240.

1 Introduction

In 1984 Karmarker presented his new polynomial-time interior-point algorithm [11], which in the past few years has led to the development of several interior point methods. For an updated bibliography on interior point methods see [13]. One of the interior point algorithms which has showed good results in practice is the affine scaling algorithm, [1,15]. The affine scaling algorithm was originally proposed in [8] and recently rediscovered in e.g. [3,20].

Consider the LP-problem:

$$(P): \quad \min_{x \in X^p} \boldsymbol{c}^T \boldsymbol{x}, \qquad X^p = \{\boldsymbol{x} \in R^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \mathbf{o}\} \tag{1}$$

where $\boldsymbol{A} \in R^{m \times n}$, $\boldsymbol{b} \in R^m$, $\boldsymbol{c} \in R^n$. The dual problem is

$$(D): \quad \max_{y \in Y^d} \boldsymbol{b}^T \boldsymbol{y}, \qquad Y^d = \{\boldsymbol{y} \in R^m \mid \boldsymbol{c} - \boldsymbol{A}^T \boldsymbol{y} = \boldsymbol{s}, \quad \boldsymbol{s} \geq \mathbf{o}\} \tag{2}$$

One virtue of the affine scaling algorithm is its simplicity. It can be viewed as replacing the simple bounds $\boldsymbol{s} \geq \mathbf{o}$ for (D) ($\boldsymbol{x} \geq \mathbf{o}$ for (P), [3]) with an ellipsoid with center in the current point $\boldsymbol{s}^0 = \boldsymbol{c} - \boldsymbol{A}^T \boldsymbol{y}^0$:

$$(DR): \quad \max\{\boldsymbol{b}^T \boldsymbol{y} \mid \boldsymbol{A}^T \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}, \quad \|S_0^{-1}(\boldsymbol{s} - \boldsymbol{s}_0)\|_2^2 \leq R^2\} \tag{3}$$

In (3) $\boldsymbol{y}^0 \in Y^{d0} = \{y \in R^m \mid \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c} \quad \boldsymbol{s} > \mathbf{o}\}$ and R is chosen such that the optimal point $\boldsymbol{y}^1$ for (3) satisfies $\boldsymbol{y}^1 \in Y^{d0}$. One step of the dual affine scaling algorithm is to find the analytic solution of the necessary first-order Karush-Kuhn-Tucker conditions for (3) in every iteration. Convergence for the dual affine scaling algorithm has been proved in [18].

In this paper we consider the dual affine scaling algorithm(DAS) as described in e.g. [1]. We define $\boldsymbol{S}_k = diag(s_1^k, \ldots, s_n^k)$ and $0 < \lambda < 1$. Let $\boldsymbol{y}^0 \in Y^{d0}$ be a given initial feasible point for (D). Then one iteration of (DAS) is:

```
$s^k \leftarrow c - A^T y^k$;
$\Delta y \leftarrow (A S_k^{-2} A^T)^{-1} b$;
$x \leftarrow S_k^{-2} A^T \Delta y$;   calculate primal variables.
$\Delta s \leftarrow -A^T \Delta y$.
$\alpha \leftarrow \min\{-s_i^k/\Delta s_i | \Delta s_i < 0, i = 1, \ldots, n\}$;
$y^{k+1} \leftarrow y^k + \lambda\alpha\Delta y$;
$k \leftarrow k + 1$;
```

As decribed above the (DAS) algorithm must start with an interior feasible point. Such a point can be found by solving an associated linear problem (phase 1). In the dual to a primal LP-problem with free variables no such interior point exists, which means that phase1 is very important: the algorithm must converge to a both feasible and optimal point. So far the "big-M" method has been the most used phase1 method in the dual algorithm ([1,15]). However, the BigM method does not work well on problems without interior points. In this paper we will motivate and develop a single phase infeasible interior point method (see e.g. [12]) which also can be seen as a dynamic big-M method (section 2). In section 3 we describe our implementation. In section 4 we present results for the algorithm on some of the problems from the netlib test-set and on a problem with a large number of free variables arising in plastic collapse analysis ([7]), for which we achieve higher accuracy and speed than done before.

2 Finding a "good" feasible (optimal) point: Phase1.

To start the DAS algorithm we must have a strong interior point to (D). We apply the DAS algorithm to a revised dual problem which converges to a strong interior point for (D). The following problem (with $M > 0$) has an optimal point which is feasible for (D) and we find a strong interior point for this problem by setting y_{ar} large enough for a given $\boldsymbol{y}$.

$$(D1)_{ar}: \quad \max\{-My_{ar} \mid \boldsymbol{A}^T\boldsymbol{y} - \boldsymbol{e}y_{ar} + \boldsymbol{s} = \boldsymbol{c}, \quad \boldsymbol{s} \geq \mathbf{o}\} \tag{4}$$

The step-direction generated by the DAS algorithm for this problem is:

$$\begin{cases} \Delta\boldsymbol{y} = \Delta y_{ar}(\boldsymbol{A}\boldsymbol{S}_k^{-2}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{S}_k^{-2}\boldsymbol{e} = \Delta y_{ar}\Delta\boldsymbol{y}_1 \\ \Delta y_{ar} = (-M + \boldsymbol{e}^T\boldsymbol{S}_k^{-2}\boldsymbol{A}^T\Delta\boldsymbol{y})/\boldsymbol{e}^T\boldsymbol{S}_k^{-2}\boldsymbol{e} \end{cases} \tag{5}$$

Only the sign of Δy_{ar}, plays a role for $\Delta\boldsymbol{y}$ because $\Delta\boldsymbol{s} = -\Delta y_{ar}(\boldsymbol{A}^T\Delta\boldsymbol{y}_1 - \boldsymbol{e})$ is merely a direction vector. We can therefore choose M so big that y_{ar} is reduced in every iteration, i.e. $\Delta y_{ar} = -1$. Then

$$\Delta\boldsymbol{y} = -(\boldsymbol{A}\boldsymbol{S}_k^{-2}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{S}_k^{-2}\boldsymbol{e} \tag{6}$$

is the feasibility direction for the dual problem. This feasibility direction is proposed in [19, pp 42-43] but they use $\boldsymbol{\rho} = \boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y} - \boldsymbol{s}$ instead of $y_{ar}\boldsymbol{e}$, which we use here.

4 Conclusion

In this note, we have presented some ideas on a data interchange tool for mathematical programming that can be used in sophisticated modeling environments to link data bases, modeling languages, report generators, and solvers.

We hope this note will serve as a basis for further discussion among those interested in the subject. In order to facilitate such a discussion the Methodology of Decision Analysis Project at IIASA is tentatively planning to organize a small workshop on data interchange tools for mathematical programming in the summer of 1993. We would like to invite researchers and software developers, interested in either developing such tools or in using them as an interface for their software. Obviously, the workshop will be organized only if a sufficient number of people are interested. The workshop should provide a useful forum for discussions and experiments that can result in improving the suggested MP-DIT specification and its implementations. For more information and other comments or suggestions please contact one of the authors.

5 References

A. Brooke, D. Kendrick, A. Meeraus (1988). *GAMS: A User's Guide*. Scientific Press, Redwood City, CA.

CPLEX Optimization, Inc. (1990). *Using the CPLEX Linear Optimizer.*

R. Fourer, D.M. Gay, B.W. Kernighan (1990). A modeling language for mathematical programming. *Management Science 36*, 519-554.

A.M. Geoffrion (1989). Computer-based modeling environments. *Europ. J. Oper. Res. 41*, 33-43.

H.J. Greenberg (1990a). *A primer for MODLER: Modeling by Object-Driven Linear Elemental Relations.* University of Colorado at Denver, Denver, CO.

H.J. Greenberg (1990b). *A primer for ANALYZE: A Computer- Assisted Analysis System for Mathematical Programming Models and Solutions.* University of Colorado at Denver, Denver, CO.

H.J. Greenberg (1990c). *A primer for RANDMOD: A System for Modifications to an Instance of a Linear Program.* University of Colorado at Denver, Denver, CO.

```
     put (int ix, ICOL *col);       /* put a single column */
get (HEADER *header);
     get (IDATA *idata);
     get (int ix, IROW *row);       /* get a single row */
     get (int ix, ICOL *col);       /* get a single column */
get (INFO *info);
     read (char *fname);            /* read an instance in
MPS-format */
     write (char *fname);           /* write an instance in MPS
format */
 };

 class solution {
     char *version;                 /* MP-DIT version */
     char *date;                    /* date */
     FILE *fp_solution;             /* disk storage */
     HEADER header;                 /* header */
     SDATA sdata;                   /* solution data */
 public:
     solution (char *fname, STORAGE storage = DISK);
     ~solution ();
     put (HEADER *header);
     put (SDATA *sdata);
     put (int ix, SROW *row);       /* put a single row */
     put (int ix, SCOL *col);       /* put a single column */
get (HEADER *header);
     get (SDATA *sdata);
     get (int ix, SROW *row);       /* get a single row */
     get (int ix, SCOL *col);       /* get a single column */
read (char *fname);             /* read a solution in MPS-format */
     write (char *fname);           /* write a solution in
MPS-format */
 };
```

However (6) does not work, if Y^{d0} is empty. Then (4) will converge to a feasible point (at the boundary), but this limit will usually not be optimal. Furthermore every iteration of the affine scaling algorithm is very expensive, so we must use each iteration as much as possible to move towards an optimal solution. Also if phase1 stops with a feasible point too close to the boundary, then many steps may be needed to find the optimal point ([16]). Hence we must start with a phase1, which also makes use of the objective function from (D). We propose the following dual problem:

$$D2_{ar} \left\{ \begin{array}{l} \max \quad (\boldsymbol{b}^T \boldsymbol{y} - M y_{ar}) \\ s.t. \\ \qquad \begin{bmatrix} A^T & -\boldsymbol{e} \\ \mathbf{o} & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{y} \\ y_{ar} \end{bmatrix} + \begin{bmatrix} \boldsymbol{s} \\ s_{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c} \\ 0 \end{bmatrix} \\ \qquad \boldsymbol{s} \geq \mathbf{o},\ s_{n+1} \geq 0. \end{array} \right. \tag{7}$$

This corresponds to the following primal problem

$$P2_{ar} \left\{ \begin{array}{l} \min \quad (\boldsymbol{c}^T \boldsymbol{x}) \\ s.t. \\ \qquad \begin{bmatrix} A & 0 \\ -\boldsymbol{e}^T & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ x_{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ -M \end{bmatrix} \\ \qquad \boldsymbol{x} \geq \mathbf{o},\ x_{n+1} \geq 0. \end{array} \right. \tag{8}$$

(7) is the same phase1 problem as proposed in [1, pp 107], except for the constraint $y_{ar} \geq 0$. It follows that the primal to their problem contains the constraint $-\boldsymbol{e}^T\boldsymbol{x} = -M$, which can be inconsistent. In (8), we only require $-\boldsymbol{e}^T\boldsymbol{x} \geq -M$. Note that when we scale y_{ar} with $y_{ar} = s_{n+1}$ we slow down the convergence for $y_{ar} \to 0$. Hence for problems with no strong interior point (then some $s_i = 0$ for all $\boldsymbol{y} \in Y^d$) we hope for a better conditioned convergence with the form (7) than with the form in [1, pp 107].

Remark 2.1 *It follows from duality theory that a necessary and sufficient condition for an optimal point $\boldsymbol{y}^*$ for $(D2_{ar})$ and $\boldsymbol{x}^*$ for $(P2_{ar})$ to be optimal also for (P) and (D) is that $M > \boldsymbol{e}^T\boldsymbol{x}^*$ and $y_{ar} = 0$.*

It follows that if a phase1 problem with a fixed M is used, then M must be chosen large enough (hence the name big-M) to ensure convergence. However this reduces the weight of the objective function, which is unsatisfactory. We conclude , therefore that M should be set dynamically.

It is interesting to compare the direction which DAS gives for $(D2_{ar})$ with the feasibility direction (6). The direction for $(D2_{ar})$ is given by

$$\left\{ \begin{array}{l} \Delta\boldsymbol{y} = (AS_k^{-2}A^T)^{-1}\boldsymbol{b} + \Delta y_{ar}(AS_k^{-2}A^T)^{-1}AS_k^{-2}\boldsymbol{e} \\ \Delta y_{ar} = (-M + \boldsymbol{e}^T S_k^{-2} A^T \Delta\boldsymbol{y})/(s_{n+1}^{-2} + \boldsymbol{e}^T S_k^{-2}\boldsymbol{e}) \end{array} \right. \tag{9}$$

The direction $\Delta\boldsymbol{y}$ is a linear combination of the feasibility direction (6) and the steepest ascent direction for the problem without the artificial variable. It follows that Δy_{ar} is the weight between the ascent direction to an optimal point and the direction to a feasible point for the dual problem (D). By setting Δy_{ar} before calculating $\Delta \boldsymbol{y}$ we can avoid the possible numeric instability that may be caused by a large M. Since y_{ar} is a measure of infeasibility of the present point, the weight $-\Delta y_{ar}$ for the feasibility direction must decrease with y_{ar}. Consider the primal problem $(P2_{ar})$ where the following equality holds:

$$-\boldsymbol{e}^T\boldsymbol{x} - x_{n+1} = -M \qquad (x_{n+1} \textit{ is a slack variabel}) \tag{10}$$

For an optimal $\boldsymbol{x}$ we have $x_{n+1} > 0$. The DAS algorithm gives the following estimate for x_{n+1} in the k'te iteration:

$$x_{n+1}^k = -\frac{\Delta y_{ar}^k}{(y_{ar}^k)^2} \tag{11}$$

Assume that $x_{n+1} = C > 0$ where C is a constant to be chosen. Then (11) reads:

$$\Delta y_{ar}^k = -\ C(y_{ar}^k)^2 \tag{12}$$

We choose the constant C scaling free:

$$C = \nu\|\boldsymbol{b}\|_\infty \tag{13}$$

where ν is a constant to be chosen (close to 1.0).

We can now propose a new dual affine scaling method, which we will call $(WDAS)$:

```
Procedure $WDAS\ (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, C, (\boldsymbol{y}^0, y_{ar}^0) \in Y^{d0}, stopping\ criterion, \lambda)$
  $k \leftarrow 0$
  do $stopping\ not\ satisfied \rightarrow$
    $\boldsymbol{s}^k \leftarrow \boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}^k + y_{ar}^k\boldsymbol{e};$
    $\Delta y_{ar} = -C(y_{ar}^k)^2;$
    $\Delta\boldsymbol{y} \leftarrow (\boldsymbol{A}\boldsymbol{S}_k^{-2}\boldsymbol{A}^T)^{-1}(\boldsymbol{b} + \Delta y_{ar}\boldsymbol{A}\boldsymbol{S}_k^{-2}\boldsymbol{e})$
    $\boldsymbol{x}^k \leftarrow \boldsymbol{S}_k^{-2}(\boldsymbol{A}^T\Delta\boldsymbol{y} - \Delta y_{ar}\boldsymbol{e});$   calculate primal variables.
    $\Delta\boldsymbol{s} \leftarrow -(\boldsymbol{A}^T\Delta\boldsymbol{y} - \Delta y_{ar}\boldsymbol{e}).$
    if $(\Delta\boldsymbol{s} \geq 0 \ \wedge\ \Delta s \neq 0)$   return unbounded;
    $\alpha \leftarrow \min\{-s_i^k/\Delta s_i | \Delta s_i < 0, i = 1, \ldots, n\};$
    $\boldsymbol{y}^{k+1} \leftarrow \boldsymbol{y}^k + \lambda\alpha\Delta\boldsymbol{y};$
    $y_{ar}^{k+1} \leftarrow y_{ar}^k + \lambda\alpha\Delta y_{ar};$
    if $y_{ar}^{k+1} \leq 0$ set $y_{ar}^{k+1} \leftarrow 0.point.$
    $k \leftarrow k + 1;$
  od
end $WDAS$.
```

Note that we have fixed $x_{n+1}^k > 0$, instead of M. It follows that

$$\boldsymbol{e}^T\boldsymbol{x}^k < M^k \tag{14}$$

and

$$y_{ar}^k \rightarrow 0\ (\Delta y_{ar}^k < 0)\ . \tag{15}$$

We see that M is chosen theoretically satisfactory in every iteration (remark 2.1). Furthermore, the BigM is avoided in the numerical factorization (practical improvement of numerical stability). For these reasons we expect the above method to perform better than the BigM method. This

```
        STATUS *rstat;                  /* row status vector */
        STATUS *cstat;                  /* column status vector */  }
SDATA;

 typedef struct {
        double val;                     /* row activity */
        double dual;                    /* dual cost */
        double slack;                   /* slack */
        STATUS status;                  /* row status */
 } SROW;

 typedef struct {
        double val;                     /* variable activity */
        double dual;                    /* reduced cost */
        STATUS status;                  /* column status */
 } SCOL;

 //
 // Classes
 //

 class instance {
        char *version;                  /* MP-DIT version */
        char *date;                     /* date */
        FILE *fp_instance;              /* disk storage */
        ACCESS access;                  /* access type */
        STORAGE storage;                /* storage type */
        HEADER header;                  /* header */
        IDATA idata;                    /* instance data */
        INFO info;                      /* instance info */
 public:
        instance (char *fname, STORAGE storage = DISK, ACCESS access
=
BYCOLS);
        ~instance ();
        put (HEADER *header);
        put (IDATA *idata);
        put (INFO *info);
        put (int ix, IROW *row);        /* put a single row */
 } IROW;

 typedef struct {
     double lbr;                    /* lower bound (-INF if no lower
bound) */
     double ubr;                    /* upper bound (INF, if no upper
bound) */
     char *cname;                   /* name */
     int nz;                        /* number of nonzero
coefficients */
     int *r;                        /* row indices of nonzero
coefficients */
     double *v;                     /* values of nonzero
coefficients */
 } ICOL;

 typedef struct {
     long iter;                     /* maximum number of iterations
*/
     long time;                     /* maximum execution time */
  double zero;                   /* zero tolerance */
     double feas;                   /* feasibility tolerance */
 //
     // This list may be extended by other commonly used parameters
and
     // tolerances
     //
 } INFO;

 //
 // Structures related to solutions to instances of linear programs
 //

 typedef struct {
     double zlp;                    /* solution value */
     double *xlp;                   /* solution vector */
     double *rc;                    /* reduced cost vector */
double *pi;                    /* dual vector */
     double *slack;                 /* slack vector */
```

was confirmed by direct comparison. We furthermore see that $(WDAS)$ is the same algorithm as (DAS) if $y_{ar}^k = 0$.

We will now show that the algorithm converges to a feasible point for (D) if (D) has a bounded feasible optimal solution. In the $WDAS$ algorithm we fix x_{n+1}. Then the following holds with C as in (12)

$$\boldsymbol{b}^T\Delta \boldsymbol{y}^k - M^k\Delta y_{ar}^k = (\boldsymbol{A}\boldsymbol{x}^k)^T\Delta \boldsymbol{y}^k - (\boldsymbol{e}^T\boldsymbol{x}^k - C)\Delta y_{ar}^k = \|S_k^{-1}\Delta \boldsymbol{s}^k\|_2^2 - C\Delta y_{ar}^k \qquad (16)$$

Theorem 2.1 *If (D) has a bounded feasible solution then $(WDAS)$ finds a feasible solution for (D): The sequence $\{\boldsymbol{y}^k, y_{ar}^k\}$ generated by $WDAS$ stops with $y_{ar}^k \leq 0$ or satisfies $\lim_{k\to\infty} y_{ar}^k = 0$.*

Proof: Assume that $\lim_{k\to\infty} y_{ar}^k \geq \delta > 0$
Then $y_{ar}^{k+1} = y_{ar}^k(1 - \alpha_k C y_{ar}^k)$ and it follows that

$$1 = \lim_{k\to\infty} \frac{y_{ar}^{k+1}}{y_{ar}^k} = \lim_{k\to\infty}(1 - \alpha_k C y_{ar}^k) \quad\Rightarrow\quad \lim_{k\to\infty} \alpha_k = 0 \qquad (17)$$

Since $\alpha_k = \min\{-((s_i^k)^{-1}\Delta s_i^k)^{-1} \mid \Delta s_i^k < 0\} \geq \|S_k^{-1}\Delta \boldsymbol{s}^k\|_\infty^{-1}$ we have:

$$\lim_{k\to\infty} \alpha_k\|S_k^{-1}\Delta \boldsymbol{s}^k\|_2^2 \geq \lim_{k\to\infty} \|S_k^{-1}\Delta \boldsymbol{s}^k\|_2 \frac{\|S_k^{-1}\Delta \boldsymbol{s}^k\|_2}{\|S_k^{-1}\Delta \boldsymbol{s}^k\|_\infty} \geq \lim_{k\to\infty} \|S_k^{-1}\Delta \boldsymbol{s}^k\|_2 \times 1 = \infty$$

From (16) we get

$$\lim_{k\to\infty} \alpha_k(\boldsymbol{b}^T\Delta \boldsymbol{y}^k - M^k\Delta y_{ar}^k) = \lim_{k\to\infty} \alpha_k\|S_k^{-1}\Delta \boldsymbol{s}^k\|_2^2 - \alpha_k\Delta y_{ar}^k C = \lim_{k\to\infty} \alpha_k\|S_k^{-1}\Delta \boldsymbol{s}^k\|_2^2 = \infty$$

Since $\lim_{k\to\infty} \alpha_k\Delta y_{ar}^k = 0$ and $\Delta y_{ar}^k < 0$, it follows that either

$$\lim_{k\to\infty} \alpha_k \boldsymbol{b}^T\Delta \boldsymbol{y}^k = \infty \qquad or \qquad \lim_{k\to\infty} M^k = \infty$$

In the first case the following dual problem is unbounded

$$\max_{y\in R^m}\{\boldsymbol{b}^T\boldsymbol{y} \mid \boldsymbol{A}^T\boldsymbol{y} \leq \boldsymbol{c} + \delta\boldsymbol{e}\} \qquad (18)$$

and it follows that the primal problem for (18) is infeasible. But then also the primal problem with $\delta = 0$ is infeasible, and from duality theory we then again have that (D) is unbounded or infeasible, which is contrary to our assumption.
In the second case it follows from $\lim_{k\to\infty} M^k = \infty$ and the convergence of (DAS), that the $(WDAS)$ algorithm must find a feasible point if there exsist one for (D). Hence (D) is infeasible, which is a contradiction. This completes the proof. □

If, at some iteration of the $WDAS$ algorithm, $y_{ar}^k \leq 0$, (note that if $y_{ar}^k = 0$ then (DAS) and $(WDAS)$ are the same algorithm), then $\boldsymbol{y}^k$ is a strong interior point for (D). Hence we only have to look at the case:

$$(S1) \qquad \lim_{k\to\infty} y_{ar}^k = 0 \quad\text{and}\quad y_{ar}^k > 0 \quad \forall k.$$

In this case we can fix y_{ar} sufficiently small and use the (DAS) algorithm to find a solution of (D), where $\boldsymbol{c}$ is slightly pertubed with $y_{ar}\boldsymbol{e}$

3 Description of the implementation

The code for the *WDAS* phase1 was programmed in *ANSI* $-$ *c*. The program reads mps format and includes implicit handling of free variables, simple bounds and slack variables.

The initial point is found as in [1, pp 306]. The algorithm is very sensitive, so the first iteration is a centering step with equal weights on the directions for feasibility and a Newton center step [15].

An estimate for the primal vector $\boldsymbol{x}$ is computed in each iteration, and the algorithm is stopped, when both primal and dual feasibility and the duality gap are less (relatively) than a given tolerance of 10^{-8}. As an emergency brake we stop if the relative improvements in the objective $\boldsymbol{b}^T\boldsymbol{y}$ gets less than another (smaller) tolerance set to 10^{-10}

Our implementation includes a preprocessor using the tecnique in [5, pp 145-150] and [17]. As mentioned in [5, pp 153] an equation with a nonzero coefficient for a free column can be eliminated along with the free column. However, this is normally not done because it will create fill-in in the constraint matrix, however if the free column is a singleton column then it can be removed along with the equations without creating fill-in in A. For such a variable we check if it is free or implied free, and in that case it is removed together with the row. This was found to give a large reduction on some problems as the following table show:

Name	Rows		Cols		Free		A		L	
capri	241	(255)	307	(321)	0	(7)	1399	(1590)	3962	(4716)
ffff800	303	(476)	646	(817)			4878	(6042)	9071	(16376)
pilot	1362	(1391)	3352	(3383)			40661	(40800)	195740	(199733)
ship12l	609	(687)	4147	(4224)			9222	(12507)	5521	(9439)

Table 1: Examples of singleton reductions. ( ) is reductions without singleton reductions.

In every iteration we have to solve a least-square and positive definite system. This was done using a supernodal Cholesky factorization similar to the one described in e.g. [10, pp 149-156] and [2]. A supernode is a set of columns in $\boldsymbol{L}$ with the same nonzero structure under the diagonal and they can therefore be handled as a dense submatrix [2, pp 17-19]. The lower corner of $\boldsymbol{L}$ will normally be dense or nearly dense. Then a dense Cholesky factorization is used for the remaing part of $\boldsymbol{L}$ if the density of a column is lager than 0.7, totally avoiding indirect addressing and therefore speeding up the algorithm significantly at the expense of a little extra storage. The use of supernodes and dense matrix techniques make it possible to unrool loops [2], this is done to a degree of 7 or 8. A slightly modified version of the minimum degree algorithm described in [10] is invoked once to minimize fill-in in $\boldsymbol{L}$.

If the dual problem is degenerate we suspect a loss of information in explicitly forming the normal-equations as we approach the optimal solution. Therefore near the solution we replace the Cholesky factorization of $\boldsymbol{AS}^{-2}\boldsymbol{A}^T$ by a QR-factorization of $(\boldsymbol{AS}^{-1})^T$ computed using Givens rotations. This could be done with the same memory and data structure [9] as for Cholesky. The code switches to QR, when Cholesky breaks down. This posibility gives on some problems a few expensive iterations, and results in a more robust and stable code.

4 Computional results.

In this section, we report the computational results of running our implementation of the WDAS (DAS) algorithm. All test was run at the CONVEX C3240. The CONVEX ANSI-c compiler was used with the "-O2"(full vectorization) option. All times are in CPU seconds and was

```
ISTATUS istatus;                 /* status of instance */      SSTATUS
sstatus;                    /* status of solution */  } HEADER;

 typedef struct {
     int obj;                        /* index of cost row */
     double *lbr;                    /* row lower bounds */
     double *ubr;                    /* row upper bounds */
     SENSE *sense;                   /* row senses */
     char **rname;                   /* row names */
     double *lbc;                    /* column lower bounds */
double *ubc;                     /* column upper bounds */      char
**cname;                    /* column names */
     int *c;                         /**/
     int *r;                         /* Storage of coefficient matrix
*/
     double *v;                      /**/
 } IDATA;

 typedef struct {
     //
     // Different data structure for the storage of the matrix,
i.e.,
     // maybe using super sparsity structures
     //
 } IDATA2;

 typedef struct {
     double lbr;                     /* lower bound (-INF if no lower
bound) */
     double ubr;                     /* upper bound (INF, if no upper
bound) */
     char sense;                     /* sense */
     char *rname;                    /* name */
     int nz;                         /* number of nonzero
coefficients */
     int *c;                         /* column indices of nonzero
coefficients */
     double *v;                      /* values of nonzero
coefficients */
```

```
*/
 #define NAMELENGTH              8    /* length of column and row names
*/
 #define PROBLEMLENGTH          20    /* length of problem name */

 #define INF              (1.e+31)    /* infinity */
 #define EPS               (1.e-6)    /* epsilon */
 #define MAXITER           1000000    /* maximum number of iterations
*/
 #define MAXTIME              3600    /* maximum number of cpu seconds
*/

 //
 // Enumeration types
 //

 enum TYPE        {MINIMIZATION, MAXIMIZATION};
 enum SENSE       {E, L, G, N, R};
 enum ACCESS      {BYCOLS, BYROWS, BYTRIPLETS};
 enum STORAGE     {MEMORY, DISK};
 enum CONTROL     {TERMINATE, SUSPEND};
 enum ISTATUS     {INITIAL, MODIFIED};
 enum SSTATUS     {FORMERRORS, UNSOLVED,
                   OPTIMAL, SUBOPTIMAL, INFEASIBLE, UNBOUNDED,
              TIMELIMIT, ITERLIMIT, NOMEMORY,
                   NUMPROBLEMS};
 enum STATUS      {BS, LL, UL, FX, FR, INFEAS};

 //
 // Structures related to instances of linear programs
 //

 typedef struct {
     char name[PROBLEMLENGTH+1];    /* problem name */
     TYPE type;                     /* minimization or maximization
*/
     int n;                         /* number of columns */
     int m;                         /* number of rows */
     long nz;                       /* number of nonzeros */
```

TITLE	ITE	PHI	CPU	PA	DA
25fv47	50	50	42	-7.4E-14	5.0E-09
80bau3b	65	65	98	-1.0E-14	1.1E-10
agg	53	53	4	-1.5E-12	-2.0E-09
agg3	28	28	6	3.0E-13	8.6E-09
bandm	28	28	3	1.2E-13	-5.7E-09
bnl1	54	54	19	1.3E-11	3.2E-09
bnl2	46	46	84	2.5E-13	3.2E-10
brandy	27	27	2	7.9E-12	3.1E-09
capri	30	30	3	3.0E-13	5.5E-09
cycle	47	47	69	1.7E-13	-1.3E-08
czprob	57	1	12	-1.7E-11	8.5E-09
d2q06c	51	51	194	-5.2E-12	2.9E-09
degen3	29	2	160	2.4E-11	-9.9E-09
e226	31	31	3	-3.5E-14	-4.0E-09
etamacro	45	45	10	-1.8E-14	-3.5E-12
fffff800	46	46	13	-6.0E-12	1.5E-09
finnis	40	40	6	-2.3E-12	3.5E-12
ganges	27	27	17	-1.1E-11	-4.9E-09
greenbea	89	89	163	1.3E-09	3.8E-09
greenbeb	63	63	117	-6.3E-12	-5.4E-12
perold	45	45	24	3.8E-08	-3.0E-09
pilot.ja	146	146	164	1.9E-03	-8.2E-07

TITLE	ITE	PHI	CPU	PA	DA
pilot.we	78	78	39	-3.3E-08	-5.8E-11
pilot	48	48	299	-3.3E-11	-4.0E-09
pilot4	79	79	30	-6.8E-08	-7.1E-08
pilot87	60	60	1092	-4.6E-14	1.1E-10
pilotnov	35	35	45	-8.7E-15	-2.5E-09
recipe	20	20	0	1.2E-14	-1.5E-09
scagr25	25	25	3	1.6E-11	-1.4E-09
scfxm3	38	38	17	-4.4E-13	2.2E-09
scrs8	51	51	8	-8.8E-15	-4.6E-09
sctap3	34	34	22	3.2E-16	3.7E-09
shell	36	36	6	-1.8E-13	2.0E-09
ship12l	26	26	10	-8.9E-14	1.7E-09
ship12s	26	26	5	-1.4E-12	6.3E-09
sierra	31	31	17	-3.9E-13	1.2E-09
stair	28	28	12	8.8E-14	-1.7E-09
standata	27	3	2	-9.0E-16	3.8E-09
stocfor2	54	54	44	-2.5E-13	1.6E-09
truss	31	0	50	-6.2E-13	4.5E-09
tuff	35	35	8	-6.8E-13	-1.8E-07
vtp.base	21	21	0	2.8E-13	2.4E-09
wood1p	31	1	41	-1.5E-11	8.1E-09
woodw	44	44	47	-4.7E-12	1.7E-08

Table 2: Results

done with the ANSI-c procedure clock. Computing time includes all processing except time for reading and converting the mps files. Table 2 presents results for some problems from the netlib test set with empty dual interior or which are known to be difficult: Ite is number of iterations, PH1 is number of iterations with $y_{ar} > 0$, CPU is computing time in seconds and PA/DA is digit accuracy of primal/dual objective value compared to the reoptimized values reported in [4, pp 10-11] (they seem to be more accurate than the values reported in the netlib index). Note the high accuracy of the primal objective value. It is further seen that the iteration count and accuracy of the final solutions are very competitive with the dual affine and primal-dual results in [1], [15] and [14], especially for problems without interior points in the dual space.

We have also tested at the problem which is described [7]. The problem is a discretization of a collapse problem. We will here only present results for N=201 (size of dicretization, see [7, pp 57]), a much finer element grid than reported earlier. The problem was dualised to get a lower row dimension. Then the problem has the following size:

$ROWS = 121204$, $COLS = 687353$, $FREE = 81338$, $|A| = 2301985$, $and\ |L| = 2301985$.
The optimizer used 98 iterations and 117099 cpu seconds. With duality gap=10^{-7}, primal feasibility=10^{-7} and dual feasibility=10^{-12} which is standard accuracy. The professional simplex optimizer Sciconic v2.11 was available at the CONVEX. For N=24 our interior point optimizer outperform Sciconic with a factor of over 130 (in the CPU time). As in [7] using the simplex optimizer MINOS we observe that the simplex optimizer has problems with finding optimal solutions. However, as pointed out in [6],with a program like MINOS one should attach the convex version of the collapse problem.

5 Conclusion.

We conclude that the method presented here is an improvement of the "fixed Big-M" method, both from a theoretical and practical point of view. For problems with many free variables it performs better than any other method, which we have tested.

Acknowledgement: I wish to thank my advisor E. Christiansen and K.O. Kortanek for their help and encourogement and for drawing my attention to the test problem in plasticity with many free variables.

Thanks also go to E.D. Andersen for valuable discussions related to interior point methods and implementation of them.

References

[1] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–235, 1989.

[2] C.C. Ashcraft, R.G. Grimes, J.G. Lewis, B.W. Peyton, and H.D. Simon. Progress in sparse matrix methods for large scale linear systems on vector supercomputers. *The International Journal of Supercomputer Applications*, 1(4):10–30, 1987.

[3] E.R. Barnes. A variation on karmarkar's algorithm for solving linear programming problems. *Mathematical Programming*, 36:174–182, 1986.

[4] R.E. Bixby. Implementating the simplex method: The initial basis. Technical Report TR90-30, Department of Mathematical Science, Rice University, 1991.

[5] G.H. Bradley, G.G. Brown, and G.W. Graves. Structurel redundancy in large-scale optmization models. In M.H. Karwan et.al., editor, *Redundancy in mathematical programming*, pages 87–109. Springer Verlag, Berlin, 1983.

[6] E. Christiansen. Limit analysis with unbounded convex yield condition. In R. Vichnevetsky and J.J.H. Miller, editors, *Proc. IMACS 13th World Congress on Computational and Applied Mathematics*, pages 129–130. Criterion Press, Dublin, 1990.

[7] E. Christiansen and K.O. Kortanek. Computation of the collapse state in limit analysis using the lp primal affine scaling algorithm. *Journal of Computional and Applied Mathematics*, 34:47–63, 1991.

[8] I.I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Math. Dokl.*, 8:674–675, 1967.

[9] A. George and M.T. Heath. Solution of sparse linear least squares problems using givens rotations. *Linear Algebra and its Applications*, 34:69–83, 1980.

[10] A. George and J.W-H.Liu. *Computing solution of large sparse positive definite systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[11] N. Karmarkar. A polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[12] M. Kojima, N. Megiddo, and S. Mizuno. A primal-dual infeasible-interior-point algorithm for linear programming. Technical report, 1991.

[13] E. Kranich. Interior point methods for mathematical programming: A bibliography. Diskussionsbeitrag Nr. 171 (Available from netlib), 1991.

[14] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and its Applications*, 20:191–222, 1991.

still open for discussion. For presentational convenience, we have restricted ourselves to data interchange for linear programs.

We like to emphasize again that the idea is to build an efficient, easy-to-use, and easy-to-extend interface tool for modeling environments.

MP-DIT can be enhanced and extended in several ways. Different data structures can be implemented for internal storage. The set of overloaded functions can be extended. New classes can be defined, for instance to support data interchange between a modeling language and report generator.

There are several issues that, for the sake of brevity, are not dealt with in the following outline, but should be considered in real implementations in order to allow the use of the MP-DIT classes with different applications and in different programming environments. Three of them are mentioned below. First, the use of `typedef` types of variables to deal with different compiler and hardware characteristics; for instance to be able to handle large problems in (MS-DOS) environments where (for many compilers) the `int` type has a size of two bytes. Second, the use of replaceable functions for memory management; in simple implementations only the operators `new` and `delete` will be used. Third, the use of replaceable functions for messages generated by MP-DIT; in simple implementations message functions will write to `stdout`.

On a different level, there are issues related to the control structure of the environment in which MP-DIT is used, i.e., the mechanisms by which the flow of execution of the various components can be specified. In a sophisticated modeling environment, for instance, there have to be mechanisms to notify both MP-DIT and the solvers (maybe via MP-DIT) that several slightly modified instances have to be solved consecutively for data modified upon analysis of obtained solutions. In such a situation, the information pertaining to the active instance should be kept for later use, even when a specific action is completed.

```
#include <stdio.h>

//
// Constant definitions
//

#define VERSION         "0.00"   /* MP-DIT version */

#define DATELENGTH          26   /* length of ascii formatted date
```

`read` and `write` for the off-line use of an object of the class. An obvious implementation of the `read` and `write` functions would be to read and write MPS files so that compatibility with the existing standard is maintained.

Another advantageous feature of C++ is the idea of function overloading. Different functions typically have different names, but for functions performing similar tasks on different types of objects it is sometimes better to let these functions have the same name. In C++ it is possible to declare a function to be overloaded, i.e., it is possible to inform the compiler that the multiple use of a function name is intentional. Using this feature it would be easy to extend the class with functions tailored to specific applications. A developer of a modeling language and data base manager may provide a specialized put function that uses implementation details of the modeling system.

To provide the necessary flexibility with respect to data manipulation, the class should provide various ways to access the instance data, such as row-wise access, column-wise access, and triplet-wise access.

To provide the necessary efficiency when used in an interactive environment the class should provide various ways to store the instance data, such as storage in memory or storage on file.

The envisioned data interchange tool can be implemented as a library of classes to be used by preprocessors, problem generators, solvers and report generators in order to provide efficient access to different sets of data processed by these modules. When used for the interchange of data between a problem generator and a solver, it should be able to handle the minimal set of data that defines an instance of a mathematical program as well as other information needed by a solver, such as different limits, options, and tolerances. When used for the interchange of data between a solver and a report generator, it should provide easy and flexible handling of a solution.

Any distribution of MP-DIT, preferably as source code, should have a basic, but fully operational, version of all functions. Each application can then replace one or more of them with tailored versions.

3 Outline of MP-DIT classes

Below we outline the basic MP-DIT classes. We do not present specific implementation details, because at this point we are mainly interested in the functionality of the tool. This section is intended to give an idea of what we have in mind; the actual implementation will depend on assumptions

[15] R.E. Marsten, M.J. Saltzman, D.F. Shanno, G.S. Pierce, and J.F. Ballintijn. Implementation of a dual affine interior point algorithm for linear programming. *ORSA Journal on Computing*, 1(4):287–297, 1989.

[16] N. Megiddo and M. Shub. Boundary behavior of interior point algorithms in linear programming. *Mathematics of Operations Research*, 14(1):97–146, 1989.

[17] J.A. Tomlin and J.S. Welch. Finding duplicate rows in a linear program. *Operations Research Letters*, 5:7–11, 1986.

[18] T. Tsuchiya. Global convergence of the affine scaling methods for degenerate linear programming problems. *Mathematical Programming*, 52:377–404, 1991.

[19] R.J. Vanderbei. Affine-scaling for linear programs with free variables. *Mathematical Programming*, 43:31–44, 1989.

[20] R.J. Vanderbei, B.A. Freeman, and M.S. Meketon. A modification of karmarkar's algorithm. *Algorithmica*, 1:395–409, 1986.

Information Utilization in Simulated Annealing and Tabu Search

Jan Paulli *

Department of Operations Research
University of Aarhus, Denmark

Abstract

Simulated annealing and tabu search have proven to be very successful heuristics for a number of combinatorial optimization problems. In this paper we will focus on the underlying principles of search in these heuristics. By introducing the notion of a "search step": a single piece of information on the structure of the problem at hand, we establish a basis for comparing simulated annealing and tabu search. An example of such a comparison on the quadratic assignment problem is given.

Keywords: Combinatorial optimization, simulated annealing, tabu search, quadratic assignment problem, information utilization.

1 Introduction

Over the past ten years simulated annealing and tabu search have proven to be very successful heuristics for a number of combinatorial optimization problems. They both are fairly easy to implement, they only require a small amount of memory space in a computer and they require very little participation of the user.

Simulated annealing and tabu search are in a sense extensions of simple local search algorithms for combinatorial optimization problems. Like simple local search algorithms they require a *starting solution* and a specification of the *neighbourhood structure*. The neighbourhood structure is the same as a *set of moves* that can be applied to any given solution to produce a new solution. A simple local search algorithm proceeds by either randomly or systematically finding a *neighbour* (a solution that can be produced by one move) that improves the objective function value and then continues the search from this neighbour. As a consequence of this myopic approach a simple local search algorithm inevitably is trapped by the first local optimum met. Simulated annealing and tabu search are in different ways trying to overcome this "trap" of local optimality.

In this paper we will focus on the underlying principles of search in simulated annealing and tabu search, not the implementational issues. We will introduce the notion of a *search step*, a single piece of information on the structure of the problem at hand and examine how a search step is done in simulated annealing and tabu search. This establishes a basis for comparing the two heuristics based on their utilization of information. An example of such a comparison on

*This work was partially supported by NorFa (Nordic Research Academy)

the description of instances of linear programming models only. Extensions have been in use for mixed integer programming models, but a more sophisticated data interchange tool should also support other type of models, such as quadratic programming, nonlinear programming, and multi-objective linear programming models. In addition to that, a sophisticated data interchange tool may even provide capabilities for the specification of structural information such as block structure in multi-period or multi-commodity flow models.

Most of the above observations and remarks are not new, but have been made before. Unfortunately, the MPS format is still the only available standard. In view of the growing importance of modeling environments and the associated challenge of software integration, the time seems right to attempt to design a viable alternative to the MPS format. This note presents some ideas on such an alternative and is intended to start a discussion among those interested in the subject.

2 Mathematical Program Data Interchange Tool (MP-DIT)

We propose, as a starting point for further discussion, an easily extendable and modifiable data interchange tool based on the class concept of C++ [Stroustrup 1986]. A class is a new data type, i.e., a concrete representation of a concept, such as an instance of a mathematical program. The fundamental idea in defining a new data type is to separate the details of implementation from the properties essential for the correct use of it. With a class, access to objects of a class can be restricted to a set of member functions declared as part of the class. One of the benefits is that a potential user of the data type need only examine the definition of the member functions to learn to use it.

We propose to define a data type that stores and manipulates instances of mathematical programs, a data type that stores and manipulates solutions to instances of mathematical programs, and possibly a data type that stores and manipulates additional information. The basic member functions that should be provided for each of these data types are put, **get**, **read**, and **write**. The put provides an object of the class with data, **get** obtains data from an object of the class, **read** gets data from a file for an object of the class, and **write** puts data in a file from an object of the class. The put and **get** functions are meant for on-line use of an object of the class and fields is very specialized, components are often built by different groups. This makes software integration a huge challenge indeed.

An important step towards software integration is the definition of standard interfaces between the components. One such interface exists: the MPS format. An MPS input file describes an instance of a mathematical programming model, an MPS output file describes a solution to an instance of a mathematical programming model, and an MPS advanced basis file describes an advanced starting point, associated with the instance described in the MPS input file, for the solution process. In fact, the only virtue of the MPS format is that it is a standard; all commercially available solvers support at least the MPS input file. However, it has many shortcomings as an interface: it is very inefficient, both in terms of time and space, and it is inflexible with respect to data manipulation.

Developing a large-scale linear programming model is an iterative process, because few modeling professionals ever get a model right the first time. Moreover, tuning a solver for a class of instances of a large-scale linear programming model is an iterative process as well, because there is a variety of methods available, such as the simplex method, the affine scaling interior point method, the primal dual interior point method, and the primal dual predictor corrector interior point method, each with numerous parameters, and it is unclear at the start which of these methods will perform best on a specific class of instances.

Envision a sophisticated environment for the development and solution of large-scale linear programming models. Such an environment may consist of a modeling language, such as GAMS [Brooke, Kendrick and Meeraus 1988], AMPL [Fourer, Gay and Kernighan 1990], or MODLER [Greenberg 1990a], one or more solvers, such as CPLEX [CPLEX 1990], OB1 [Lustig, Marsten, and Shanno 1989, 1990], or OSL [IBM 1991], an interactive system to provide computer assisted analysis of model instances and their solution, such as ANALYZE [Greenberg 1990b], and a tool to generate random variations around a base model for statistical experimentation, such as RANDMOD [Greenberg 1990c].

In such an environment there is a lot of communication between the various components. MPS files are not very well suited to provide such a means of communication. They are inflexible and inefficient. The overhead of writing and reading a complete MPS file, even if a previously generated and solved instance is only slightly modified, is enormous. What is needed is a more flexible and efficient data interchange tool.

Another disadvantage of the MPS format is that it was designed for

the quadratic assignment problem is given. We hope by this to be able to clarify the similarities and the differences between simulated annealing and tabu search.

In what follows we will be considering the minimization case. This is done to avoid ambiguity in language and the maximization case can be obtained by exchanging words like "increase" with "decrease".

2 Simulated annealing

The basic idea in simulated annealing is to choose a neighbour at random and then move to this neighbour if either it has a better (lower) objective function value or, in case the neighbour has a higher objective function value, if $\exp(-\frac{\Delta}{T}) \geq U$. Δ is the increase in objective function value if we move to the neighbour, T is a control parameter, that is reduced during the search and U is a random number between 0 and 1. In other words, we always move to a better neighbour if we find one and sometimes we also move to a neighbour with a higher objective function value. However the bigger the increase in objective function value the smaller the probability of accepting the move. The effect of decreasing T is that the probability of accepting an increase in the objective function value is decreased during the search.

The tricky part of simulated annealing, apart from finding a suitable neighbourhood structure, is to decide what initial value of T to choose and how T should decrease during the search. Though simulated annealing is shown to converge to optimality if T is controlled in the right way (see e.g. [8]) the rate of convergence is too slow. Consequently one has to choose some compromise between accuracy and speed. For more details on simulated annealing see e.g. [8] and [15].

3 Tabu search

As opposed to the randomness of the search in simulated annealing, tabu search uses a more systematic approach. Instead of choosing a neighbour at random, tabu search examines all of the neighbours and selects the best *admissible move* away from the current solution. An admissible move is a move, that is not on the *tabu list*, where the tabu list is a list containing forbidden moves. Normally the forbidden moves are the reverse of the s moves last performed. The tabu list is initialized empty, constructed in the s first iterations and updated circularly in later iterations. The parameter s is called the *tabu list size*.

If the current solution is a local minimum there is no improving moves and in this case the best admissible move is a move that increases the objective function value as little as possible. Due to the tabu list tabu search avoids returning to the solutions just visited, and since degrading moves are allowed, the algorithm has a chance of leaving a local minimum. Usually one introduces an *aspiration level criterion* to speed up the search. That is, we allow a move currently on the tabu list if it leads to a better solution than the best found so far.

The most difficult part of applying tabu search is finding the right size of s. If the tabu list size is too small the search will start cycling and if it is too large the search might be too restrictive. Again it is a question of finding the right compromise, typically depending on the specific problem investigated. For more details on tabu search see e.g. [5] and [6].

4 Comparing simulated annealing and tabu search

Note, that each time a neighbour is considered the algorithms are given (local) information about the structure of the problem. Simulated annealing uses this information to decide whether to go to this neighbour or not, whereas tabu search uses it to build a picture of the neighbourhood by storing the information and examining the rest of the neighbours before deciding on a move. So tabu search uses the same amount of information to perform one move as simulated annealing uses in N moves. Here N is the number of neighbours of a given solution. On the other hand, it is quite possible that simulated annealing by chance has chosen N neighbours with a higher objective function value and with $\exp(-\frac{\Delta}{T}) < U$ and consequently has not moved at all. Put differently, simulated annealing might decide not to use N pieces of information on the problem structure, whereas tabu search always uses this amount of information to perform one move. We define a *search step* as one evaluation of a neighbour (or equivalently, one piece of information on the structure of the problem). Thus tabu search uses N search steps to make one move, whereas simulated annealing uses an unknown number (larger than or equal to one) of search steps to perform one move.

One might characterize tabu search as a thorough but slow moving search and simulated annealing as a fast moving but cursory search. As this is two quite opposite philosophies of search an obvious question is: Which kind of search is the best?

This question has been addressed quite a few times during the past years (see e.g. [11], [13], [14] and [10]). Unfortunately the answer is not that simple. In most cases there will be an obvious trade-off between solution quality and speed, making it difficult to compare different methods. Furthermore there are other aspects to be considered like ease of implementation and storage requirements. Finally the answer will also depend a great deal on what kind of problems we want to examine. Consequently we will usually have to settle for an evaluation on a limited number of problems in a given setting. This paper is no exception.

Our idea is now to examine how well simulated annealing utilizes information about the problem structure compared to tabu search. Given the same problems and starting solutions we will examine the solution value after the same number of search steps using either simulated annealing or tabu search. Obviously we would expect to find that simulated annealing is best at the beginning, as it visits more solutions than tabu search. But sooner or later we would expect the systematical approach of tabu search to gain on simulated annealing. The interesting question is, how long this is going to take.

As previously mentioned we will have to evaluate only a limited number of problems. In this paper we will be considering instances of the quadratic assignment problem. The reason for choosing the quadratic assignment problem is the fact that this problem is easy to state but hard to solve. Furthermore the application of simulated annealing and tabu search for the quadratic assignment problem has received quite of lot of attention in the literature (see e.g. [4], [11], [13], [1] and [10])

Before we start our comparison we will give a short presentation of the quadratic assignment problem. For a more thorough presentation of the quadratic assignment problem and an extensive list of different solution methods, exact and heuristic, see [2].

4.1 The quadratic assignment problem

In simple terms the purpose of the quadratic assignment problem minimize the costs attached to the assignment of n facilities to n locations, when the costs depend on both distance and

MP-DIT
Mathematical Program Data Interchange Tool

Marek Makowski
International Institute for Applied Systems Analysis
2361 Laxenburg
Austria

Martin W.P. Savelsbergh
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

1 Introduction

In a paper on modeling environments Geoffrion [1989] states that 'modeling environments have the potential to greatly increase the productivity of model-based work through better tools, to improve the quality of model-based work through better support for good modeling style and work practices, and to improve the frequency of use of MS/OR by bringing about a more comfortable working relationship between MS/OR professionals and their constituencies.' Geoffrion argues that 'facilities for good management of key resources, namely data, models, solvers, and knowledge derived from these' is one of the desired characteristics of a modeling environment and that 'software integration' is one of the major challenges for the future.

The three major components of a modeling environment are a *data base*, see Welch [1987] for a discussion on the data management needs for mathematical programming applications, a *modeling language*, see Tomlin and Welch [1993] for a historic review of the developments in the area of matrix generators and modeling languages, and *solvers*. Over the years most research has been devoted to solvers, with a growing interest in modeling languages. The data base component, although maybe the most important from a practical point of view, has received the least attention from the mathematical programming community. Furthermore, since each of these

Table 2: Problem Origins.

NAME	ORIGINATOR	FORMULATOR	DONATOR
air01-air06			Greg Astfalk
bell3a	William Cook	William Cook	William Cook
bell3b	William Cook	William Cook	William Cook
bell4	William Cook	William Cook	William Cook
bell5	William Cook	William Cook	William Cook
bm23	B. Bouvier, G. Messoumain	Harlan Crowder	E. Andrew Boyd
cracpb1	Harlan Crowder	Harlan Crowder	E. Andrew Boyd
diamond	John W. Gregory	John W. Gregory	E. Andrew Boyd
dsbmip			John J. Forrest
egout		Laurence A. Wolsey	Martin W. P. Savelsbergh
enigma	Harlan Crowder	Harlan Crowder	E. Andrew Boyd
fixnet3		Laurence A. Wolsey	Martin W. P. Savelsbergh
fixnet4		Laurence A. Wolsey	Martin W. P. Savelsbergh
fixnet6		Laurence A. Wolsey	Martin W. P. Savelsbergh
flugpl	Harvey M. Wagner	John W. Gregory	E. Andrew Boyd
gen		Laurence A. Wolsey	Martin W. P. Savelsbergh
khb05250		Laurence A. Wolsey	Martin W. P. Savelsbergh
l152lav	Harlan Crowder	Harlan Crowder	John W. Gregory
lp4l	Harlan Crowder	Harlan Crowder	E. Andrew Boyd
lseu	C. E. Lemke, K. Spielberg	Ellis L. Johnson, Uwe H. Suhl	John J. Forrest
modglob		Laurence A. Wolsey	Martin W. P. Savelsbergh
misc01-misc07			Greg Astfalk
mod008	IBM France	IBM France	John J. Forrest
mod010	IBM Yorktown Hts	IBM Yorktown Hts	John J. Forrest
mod011	Uwe H. Suhl	Uwe H. Suhl	John J. Forrest
mod013	Laurence A. Wolsey	Laurence A. Wolsey	John J. Forrest
mps_format			Shireen Sara Dadmehr
noswot		Linus E. Schrage	John W. Gregory
p0033	CJP set		E. Andrew Boyd
p0040	CJP set		E. Andrew Boyd
p0201	CJP set		E. Andrew Boyd
p0282	CJP set		E. Andrew Boyd
p0291	CJP set		E. Andrew Boyd
p0548	CJP set	Ellis L. Johnson	E. Andrew Boyd
p2756	CJP set	Ellis L. Johnson	E. Andrew Boyd
p6000	Karla Hoffman, Manfred Padberg	Telecommunications Corporation	Karla Hoffman
pipex		Laurence A. Wolsey	Martin W. P. Savelsbergh
rentacar			John J. Forrest
rgn		Laurence A. Wolsey	Martin W. P. Savelsbergh
sample2		Laurence A. Wolsey	Martin W. P. Savelsbergh
sentoy		Senju-Toyoda	Linus E. Schrage
set1al		Laurence A. Wolsey	Martin W. P. Savelsbergh
set1ch		Laurence A. Wolsey	Martin W. P. Savelsbergh
set1cl		Laurence A. Wolsey	Martin W. P. Savelsbergh
stein15	George L. Nemhauser	John W. Gregory	E. Andrew Boyd
stein27	George L. Nemhauser	John W. Gregory	E. Andrew Boyd
stein45	George L. Nemhauser	John W. Gregory	E. Andrew Boyd
stein9	George L. Nemhauser	John W. Gregory	E. Andrew Boyd
vpm1		Laurence A. Wolsey	Martin W. P. Savelsbergh

interaction between facilities. More formally, let f_{ik} denote the cost of moving the flow between facility i and facility k one unit of distance and let d_{jl} denote the distance between location j and location l. We want to assign the facilities to the locations, such that the total flow cost is minimized. An assignment of n facilities to n locations is equivalent to a permutation φ of $\{1,\ldots,n\}$ and the quadratic assignment problem can now be formulated as:

$$\min_{\varphi\in\Phi}\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ik}d_{\varphi(i)\varphi(k)}, \quad \text{where } \Phi \text{ is the set of all permutations } \varphi. \tag{1}$$

At present it seems impossible to solve a quadratic assignment problem with arbitrary coefficients exactly for $n > 18$ (see [7]) and simulated annealing and tabu search is by far the most successful heuristics.

The easiest way to define a neighbourhood structure for the quadratic assignment problem is to define a move as an exchange of two assignments in a given solution. A neighbour $\overline{\varphi}$ of a solution φ is thus of the form:

$$\overline{\varphi}(i) = \varphi(k),\ \ \overline{\varphi}(k) = \varphi(i)\ \ and\ \ \overline{\varphi}(l) = \varphi(l),\ \ l \neq i,k \tag{2}$$

It is easy to see that all solutions can be obtained using the above defined moves and neighbourhood structure. N, the number of neighbours of a given solution, is equal to $\frac{n(n-1)}{2}$.

4.2 Outline of experiment

Simulated annealing and tabu search are given the same starting solutions constructed at random. The algorithms are each performing M search steps from the starting solution and the best solution found during the search is recorded. In our experiments we used M so that $\frac{M}{N} =$ 1, 5, 10, 25, 50, 100, 200, 400, 800 and 1600.

We have chosen a rather simple way of controlling the parameter T in our implementation of simulated annealing. The method were originally suggested in [4] and later improved in [10]. Before the search was started, we randomly selected a small number (here: $\frac{N}{4}$) of neighbours to the starting solution and determined Δ_{max} and Δ_{min}, the maximal and the minimal increase in the objective function value. The initial value of T was now set equal to $\Delta_{min}+\frac{1}{10}(\Delta_{max}-\Delta_{min})$ and T was decreased after each step by $\frac{1}{10}(\Delta_{max}-\Delta_{min})/M$. In this way T equals Δ_{min} at the end of the search.

In order to avoid a lot of fine-tuning of the tabu list size s we adopted a strategy proposed in [13]. Instead of having a fixed size s was changed randomly between s_{max} and s_{min} every $2Ns_{max}$ search steps. We used $\lfloor 0.9n \rfloor$ and $\lceil 1.1n \rceil$ for s_{min} and s_{max} in accordance with [13]. Our tabu list size thus fluctuated round n, the size of the problem.

The data used in the experiments were: The four largest Nugent problems (Nug12, Nug15, Nug20 and Nug30) [9], the Steinberg-problem with rectangular, Euclidean and squared Euclidean distances (Ste1, Ste2 and Ste3) [12], Burkard and Offermann's German electrical typewriter data (Ger26) [3], a Danish electrical typewriter data sample (Den29, own data - not published), three 30x30 problems (Ran1, Ran2 and Ran3) and three 50x50 problems (Ran4, Ran5 and Ran6). The six random problems were constructed using a 5x6 and a 5x10 grid with rectangular distances and uniform numbers on [0,100] for costs. However in order to give the problems different complexities the costs were set equal to zero with probability $\frac{1}{3}$ in Ran2 and Ran5 and with probability $\frac{2}{3}$ in Ran3 and Ran6.

4.3 Results

The results of our experiments are shown in table 1 and 2. The algorithms were given 100 different starting solutions for each problem and the average deviations in percent from the best known solution were found.

Problem	Size	Best known solution	Avg. dev. in % from best known solution when M/N equals									
			1	5	10	25	50	100	200	400	800	1600
Nug12	12	578	11.64	5.46	3.97	2.57	2.05	1.44	0.89	0.53	0.26	0.10
Nug15	15	1150	11.94	4.49	2.76	1.65	1.13	0.68	0.50	0.22	0.12	0.04
Nug20	20	2570	9.63	4.33	3.33	2.14	1.69	1.18	0.76	0.47	0.35	0.16
Nug30	30	6124	8.93	4.18	2.90	1.85	1.35	0.99	0.79	0.57	0.38	0.30
Ger26	26	381985.5	1.44	0.59	0.41	0.24	0.22	0.17	0.16	0.15	0.12	0.12
Den29	29	6734102	1.06	0.43	0.31	0.21	0.17	0.13	0.11	0.07	0.07	0.05
Ste1	36	4763	32.59	16.15	11.04	8.08	6.03	3.95	3.05	2.56	1.89	1.45
Ste2	36	4119.75	27.81	13.21	9.69	6.57	4.25	3.31	2.21	1.65	1.04	0.82
Ste3	36	7926	85.41	41.20	30.22	19.22	13.15	9.33	6.79	4.79	3.69	2.58
Ran1	30	140662	3.44	1.53	1.13	0.77	0.62	0.45	0.32	0.22	0.18	0.11
Ran2	30	82740	8.11	3.79	2.71	1.88	1.48	1.07	0.81	0.60	0.34	0.20
Ran3	30	34574	15.23	7.36	5.52	3.46	2.78	2.19	1.66	1.09	0.84	0.53
Ran4	50	541894	2.73	1.26	0.85	0.55	0.40	0.29	0.24	0.20	0.14	0.12
Ran5	50	323876	5.41	2.65	1.91	1.34	1.02	0.82	0.66	0.54	0.36	0.32
Ran6	50	149358	10.85	5.64	3.96	2.49	1.91	1.43	0.97	0.75	0.54	0.42

Table 1: Average solution quality for simulated annealing using 100 different starting solutions.

Problem	Size	Best known solution	Avg. dev. in % from best known solution when M/N equals									
			1	5	10	25	50	100	200	400	800	1600
Nug12	12	578	25.08	7.14	3.63	2.04	1.54	0.92	0.61	0.16	0.03	0.00
Nug15	15	1150	26.55	9.41	4.22	2.02	0.94	0.55	0.26	0.12	0.05	0.03
Nug20	20	2570	24.98	11.20	5.73	2.94	1.88	1.27	0.73	0.49	0.24	0.11
Nug30	30	6124	27.74	16.03	9.68	4.36	2.60	1.85	1.20	0.78	0.49	0.27
Ger26	26	381985.5	6.73	1.99	0.69	0.32	0.30	0.29	0.29	0.29	0.29	0.29
Den29	29	6734102	7.80	2.56	0.69	0.20	0.20	0.19	0.19	0.18	0.18	0.18
Ste1	36	4763	106.10	60.54	36.29	14.58	10.13	7.98	6.07	4.74	3.86	2.99
Ste2	36	4119.75	97.16	54.79	31.53	11.86	7.77	5.73	4.44	3.62	2.90	2.23
Ste3	36	7926	313.31	156.69	85.57	29.37	17.90	13.50	10.60	8.61	7.08	5.67
Ran1	30	140662	11.23	6.33	3.62	1.48	0.91	0.63	0.41	0.25	0.13	0.07
Ran2	30	82740	23.61	13.67	8.22	3.75	2.55	1.96	1.46	1.06	0.65	0.30
Ran3	30	34574	46.00	26.14	16.00	7.37	4.91	3.56	2.34	1.32	0.72	0.44
Ran4	50	541894	10.31	7.02	4.84	2.36	1.34	0.94	0.72	0.55	0.40	0.26
Ran5	50	323876	20.08	13.61	9.41	4.42	2.59	1.81	1.48	1.13	0.87	0.68
Ran6	50	149358	38.25	26.43	18.70	9.43	5.70	4.12	3.12	2.35	1.78	1.42

Table 2: Average solution quality for tabu search using 100 different starting solutions.

To help compare table 1 and 2, table 3 shows the results for simulated annealing deducted from the results for tabu search. That is, the numbers in table 3 are the difference between

Table 1: Problem Statistics.

NAME	ROWS	COLS	INT	0/1	INT SOLUTION	LP SOLUTION
air01	23	771	771	ALL	6796 (opt)	6743.0
air02	50	6774	6774	ALL	7810 (opt)	7640.0
air03	124	10757	10757	ALL	340160 (opt)	338864.25
air04	823	8904	8904	ALL	56137 (opt)*	55535.436
air05	426	7195	7195	ALL	26374 (opt)*	25877.609
air06	825	8627	8627	ALL	49649 (opt)	49616.364
bell3a	123	133	71	39	878430.32 (opt)*	862578.64
bell3b	123	133	71	39	11786160.62 (opt)*	11404143.89
bell4	105	117	64	34	18541484.20 (opt)*	17984775.91
bell5	91	104	58	30	8966406.49 (opt)*	8608417.95
bm23	20	27	27	ALL	34 (opt)	20.57
cracpb1	143	572	572	ALL	22199 (opt)	22199.0
diamond	4	2	2	ALL	integer infeasible	-1.0
dsbmip	1182	1886	192	160	-305.198 (opt)	-305.198
egout	98	141	55	ALL	568.101 (opt)*	149.589
enigma	21	100	100	ALL	0.0 (opt)	0.0
fixnet3	478	878	378	ALL	51973 (opt)*	40717.018
fixnet4	479	878	378	378	8936 (opt)*	4257.97
fixnet6	479	878	378	378	3983 (opt)*	1200.88
flugpl	18	18	11	0	1201500 (opt)	1167185.73
gen	780	870	150	144	112313 (opt)*	112130.0
khb05250	101	1350	24	ALL	106940226 (opt)	95919464.0
l152lav	97	1989	1989	ALL	4722 (opt)	4656.36
lp4l	85	1086	1086	ALL	2967 (opt)	2942.5
lseu	28	89	89	ALL	1120 (opt)	834.68
modglob	291	422	98	ALL	20740508 (opt)*	20430947.0
misc01	54	83	82	ALL	563.5 (opt)	57.0
misc02	39	59	58	ALL	1690 (opt)	1010.0
misc03	96	160	159	ALL	3360 (opt)	1910.0
misc04	1725	4897	30	ALL	2666.699 (opt)	2656.42
misc05	300	136	74	ALL	2984.5 (opt)	2930.9
misc06	820	1808	112	ALL	12850.8607 (opt)	12841.69
misc07	212	260	259	ALL	2810 (not opt)	1415.0
mod008	6	319	319	ALL	307 (opt)	290.93
mod010	146	2655	2655	ALL	6548 (opt)	6532.08
mod011	4480	10958	96	ALL	-54558535 (opt)*	-62121982.552
mod013	62	96	48	ALL	280.95 (opt)	256.02
noswot	182	128	100	75	-43 (opt)	-43.0
p0033	16	33	33	ALL	3089 (opt)	2520.57
p0040	23	40	40	ALL	62027 (opt)	61796.55
p0201	133	201	201	ALL	7615 (opt)	6875.0
p0282	241	282	282	ALL	258411 (opt)	176867.50
p0291	252	291	291	ALL	5223.7490 (opt)	1705.13
p0548	176	548	548	ALL	8691 (opt)	315.29
p2756	755	2756	2756	ALL	3124 (opt)*	2688.75
p6000	2095	5872	5872	5872	-2350544 (opt)*	-2351871.325
pipex	25	48	48	ALL	788.263 (opt)	773.751
rentacar	6803	9557	55	ALL	30356761 (opt)*	28806137.644
rgn	24	180	100	ALL	82.1999 (opt)	48.7999
sample2	45	67	21	ALL	375 (opt)	247.0
sentoy	30	60	60	ALL	-7772 (opt)	-7839.278
set1al	493	712	240	240	15869.7 (opt)*	11145.6
set1ch	493	712	240	240	54537.7 (opt)*	32007.7
set1cl	493	712	240	240	6484.25 (opt)*	1671.96
stein15	36	15	15	ALL	9 (opt)	7.0
stein27	118	27	27	ALL	18 (opt)	13.0
stein45	331	45	45	ALL	30 (opt)*	22.0
stein9	13	9	9	ALL	5 (opt)	4.0
vpm1	234	378	168	ALL	20 (opt)*	15.4167

MIPLIB users will allow this file to be expanded.

4 Accessing MIPLIB

MIPLIB is accessible through SOFTLIB, an electronic software distribution system available through Internet. By sending e-mail to softlib@rice.edu with a message body reading "send catalogue," you will receive via electronic mail a list of software codes and libraries currently available for public access through the system, along with specific instructions for accessing each one.

In addition to making this library available for public use, we wish to invite comments and suggestions regarding the library. Contributions of new problems to the library would also be greatly appreciated, as well as any new solutions that have been proven optimal.

To make submissions or comments, please contact Dr. Robert E. Bixby, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77251, or Dr. E. Andrew Boyd, Department of Industrial Engineering, Texas A&M University, College Station, TX, 77843-3131, or send e-mail to either bixby@rice.edu or boyd@marvin.tamu.edu.

References

[1] Bixby, R. E., E. A. Boyd, and R. R. Indovina. 1992. MIPLIB: A Test Set of Mixed Integer Programming Problems. *SIAM News* **25**:2 (March).

[2] Gay, D. M. 1985. Electronic Mail Distribution of Linear Programming Test Problems. *COAL Newsletter* **13** (December) 10-12.

[3] Reinelt, G. 1991. TSPLIB — A Traveling Salesman Problem Library. *ORSA Journal on Computing* **3**:4, 376-384.

the average solution quality of tabu search and of simulated annealing relative to best known solution.

Problem	Size	Avg. dev. in % from best known solution when M/N equals									
		1	5	10	25	50	100	200	400	800	1600
Nug12	12	13.4	1.7	-0.3	-0.5	-0.5	-0.5	-0.3	-0.4	-0.2	-0.1
Nug15	15	14.6	4.9	1.5	0.4	-0.2	-0.1	-0.2	-0.1	-0.1	0.0
Nug20	20	15.3	6.9	2.4	0.8	0.2	0.1	0.0	0.0	-0.1	-0.1
Nug30	30	18.8	11.8	6.8	2.5	1.2	0.9	0.4	0.2	0.1	0.0
Ger26	26	5.3	1.4	0.3	0.1	0.1	0.1	0.1	0.1	0.2	0.2
Den29	29	6.7	2.1	0.4	0.0	0.0	0.1	0.1	0.1	0.1	0.1
Ste1	36	73.5	44.4	25.2	6.5	4.1	4.0	3.0	2.2	2.0	1.5
Ste2	36	69.4	41.6	21.8	5.3	3.5	2.4	2.2	2.0	1.9	1.4
Ste3	36	227.9	115.5	55.4	10.2	4.8	4.2	3.8	3.8	3.4	3.1
Ran1	30	7.8	4.8	2.5	0.7	0.3	0.2	0.1	0.0	0.0	0.0
Ran2	30	15.5	9.9	5.5	1.9	1.1	0.9	0.7	0.5	0.3	0.1
Ran3	30	30.8	18.8	10.5	3.9	2.1	1.4	0.7	0.2	-0.1	-0.1
Ran4	50	7.6	5.8	4.0	1.8	0.9	0.6	0.5	0.3	0.3	0.1
Ran5	50	14.7	11.0	7.5	3.1	1.6	1.0	0.8	0.6	0.5	0.4
Ran6	50	27.4	20.8	14.7	6.9	3.8	2.7	2.2	1.6	1.2	1.0

Table 3: The difference between the average solution quality for tabu search and simulated annealing.

The results confirm our expectations. In the beginning before tabu search has sufficient information to escape from a possible bad starting solution simulated annealing is much better than tabu search. But as the search goes on the differences between the the solution qualities are reduced. In most case tabu search catches up with simulated annealing after $400N - 800N$ search steps. The only exceptions are the three Steinberg problems. This reflects the fact, that these problems are extremely difficult, and thus tabu search requires more information. The same can be observed on the random problems, where increased complexity increases the differences between simulated annealing and tabu search.

Another interesting observation is that at the time tabu search catches up with simulated annealing the solution quality is usually within $0.5 - 1.0\%$ of the best known solution. This leaves very little room for tabu search to outperform simulated annealing. Based on these observations it seems that the simulated annealing approach is the best way to use (local) information on the structure of a quadratic assignment problem. In other words, fast and cursory is better that thorough and slow.

5 Final remarks and future extensions

In this paper we have been comparing simulated annealing and tabu search as two opposite ways of using gained information about the structure of a complex problem. Tabu search was characterized as a thorough but slow way of using the information whereas simulated annealing is a fast but cursory way. In order to investigate the relationship between these two philosophies of search, we compared their performance on 15 instances of the quadratic assignment problem. We found that simulated annealing seems to be using the information most efficiently.

It should be noted, that we do not claim that simulated annealing is the best heuristic for the quadratic assignment problem, only that simulated annealing does not have to investigate as many neighbours as tabu search to achieve a reasonable solution quality.

Implementational issues like data structures having a severe impact on the speed of the search have not been considered in this paper. This is part of the implementation optimization problem and is thus extremely dependent on the problem considered and the skills of the programmer. Consequently the efficiency of information utilization seems to be a useful guide in evaluating heuristics.

In this paper our experiments have been limited to the quadratic assignment problem. Examples of other areas where simulated annealing and tabu search have been successful and where a comparison might be interesting are flow and job shop scheduling and the travelling salesman problem. An obvious extension is thus to investigate whether the results found here can be extended to other problems.

References

[1] Andersen, K. and R. V. V. Vidal, "Solving the Quadratic assignment Problem", in R. V. V. Vidal (ed.), *Applied Simulated Annealing* (1993) pp. 61-83, Springer Verlag.

[2] Burkard, R. E., "Locations with Spatial Interactions: The Quadratic Assignment Problem", in Mirchandani, P. B. and R. L. Francis (eds.), *Discrete Location Theory* (1990) Wiley Interscience.

[3] Burkard, R. E. and J. Offermann, "Entwurf von Schreibmaschinentastaturen Mittels Quadratischer Zuordnungsprobleme", *Zeitschrift für Operations Research* 21 (1977) pp. B121-B132.

[4] Connolly, D. T., "An Improved Annealing Scheme for the QAP", *EJOR* 46 (1990) pp. 93-100.

[5] Glover, F., "Tabu Search-Part I", *ORSA Journal on Computing* 1, 3 (1989) pp. 190-206.

[6] Glover, F., "Tabu Search-Part II", *ORSA Journal on Computing* 2, 1 (1990) pp. 4-32.

[7] Laursen, P. S., "New Parallel Branch and Bound Algorithms for the Quadratic Assignment Problem", *master thesis* (1991) Department of Computer Science, University of Copenhagen.

[8] Lundy, M. and A. Mees., "Convergence of an Annealing Algorithm", *Mathematical Programming* 34 (1986) pp. 111-124.

[9] Nugent, C. E., T. E. Vollmann and J. Ruml, "An experimental Comparison of Techniques for the Assignment of Facilities to Locations", *Journal of Operations Research* 16 (1968) pp. 150-173.

[10] Paulli, J., "A Computational comparison of Simulated Annealing and Tabu Search Applied to the Quadratic Assignment Problem", in R. V. V. Vidal (ed.), *Applied Simulated Annealing* (1993) pp. 85-102, Springer Verlag.

[11] Skorin-Kapov, J., "Tabu Search Applied to the Quadratic Assignment Problem", *ORSA Journal on Computing*, 2, 1 (1990) pp. 33-45.

[12] Steinberg, L., "The Backboard Wiring Problem : A Placement Algorithm", *SIAM Review* 3 (1960) pp. 37-50.

[13] Taillard, E., "Robust Taboo Search for the Quadratic Assignment Problem", *Parallel Computing* 17 (1991) pp. 443-455

[14] Taillard, E., "Parallel taboo search techniques for the job shop scheduling problem", *Internal Report ORWP 89/11, Departement de mathematiques, Ecole Polytechniques Federale de Lausanne, Lausanne*, 1989. To appear in ORSA Journal on Computing.

[15] Vidal, R. V. V. (ed.), *Applied Simulated Annealing* (1993) Springer Verlag.

information regarding the origins of each problem. In Table 1, the first column, NAME, contains the name of the model. The next two columns, ROWS and COLS, contain the number of rows in the constraint matrix, not including free rows, and the number of variables in the problem, respectively. The column INT indicates the number of variables that are restricted to integer values, and the column 0/1 indicates how many of these integer variables are binary. The last two columns report the optimal solution value and the optimal solution value with the integrality restrictions relaxed, respectively. In the column INT SOLUTION however, the solutions reported are followed by qualifiers. The qualifier (opt) indicates that the reported solution is indeed integer optimal, while (not opt) indicates that the reported solution is not integer optimal. The qualifier (opt)* indicates that the solution is reportedly integer optimal, as stated by various sources, but it has not yet been verified by us.

The first column of Table 2 is, again, NAME. The second column, ORIGINATOR, gives the name of the person or institution with which the problem originated. The next column, FORMULATOR, gives the name of the person or organization responsible for formulating the model, and the last column, DONATOR, gives the name of the person or institution who contributed the problem. Much of the credit for MIPLIB resides with these individuals.

3 Modifications

In response to donations and comments by users, modifications have recently been made to MIPLIB: six new problems have been added, solution values of two problems have been improved upon due to tolerance resetting, an optimal solution has been found for a previously unsolved problem, documentation on MPS format has been included, and a list of articles which make reference to various MIPLIB problems has been added.

Dr. Karla Hoffman and Dr. Martin Savelsbergh have been kind enough to donate new problems to be used in MIPLIB. Dr. Hoffman has donated p6000, and Dr. Savelsbergh has donated fixnet4, fixnet6, set1al, set1ch, and set1cl. Further information about these problems can be found in Tables 1 and 2. In addition, Dr. Hoffman has informed us that the solutions to problems air04 and air05 have been improved upon due to tolerance resetting; Dr. Applegate, Dr. Bixby, and Dr. Cook have found an optimal solution for l152lav. These new solutions are also recorded in Table 1.

The file mps_format has been added to MIPLIB to serve as a brief introduction and guideline to MPS formatting. The MPS file format was originally introduced by IBM to express linear and integer programs in a standard way. The format is a fixed column format, so care must be taken that all information is placed in the correct columns. All models in MIPLIB are written in separate files in MPS format.

Several MIPLIB users have expressed an interest in knowing if articles exist which make reference to any MIPLIB problems. For this reason, the file *references* has been added to the library. This document contains a list of various MIPLIB problems and articles which make reference to them. It is hoped that information supplied by

The MIPLIB Mixed Integer Programming Library

Robert E. Bixby E. Andrew Boyd Shireen S. Dadmehr
Ronni R. Indovina

October 19, 1992

1 Introduction

Mathematical programs arising from real applications almost uniformly contain structure and sparsity not found in their randomly generated counterparts. These differences are often so significant that algorithms developed for randomly generated problems are almost guaranteed to perform poorly on the real problems they were presumably intended to solve.

The need for problems arising from real applications was recognized early in the development of linear programming algorithms. In 1985, an electronically accessible library of linear programming problems was introduced which has become a widely recognized foundation for computational work in linear programming [2]. The library consists of a collection of problems contributed for the most part by practitioners. The library continues to be modified, and most recently some significantly larger problems have been added in response to the need of developers to stretch the limits of their algorithms. A similar library was introduced in 1990 for research on traveling salesman problems [3]. As a result of the introduction of the traveling salesman library, 13 previously unsolved problems have now, in fact, been solved.

While mixed integer programs are used extensively to model real applications, no similar library has ever been created for use by developers of mixed integer programming algorithms until very recently. MIPLIB is an electronic library of integer and mixed integer programs created in an effort to fill this void. MIPLIB was originally developed by three of the authors and described in [1]. The purpose of this paper is to provide pertinent information on MIPLIB and to describe recent changes and modifications which have been made to the library.

2 MIPLIB

Tables 1 and 2 contain information about the models currently contained in MIPLIB. The first table contains statistics for each problem, and the second table contains

Editorial

This issue of the COAL Bulletin consists of four papers, two dealing with data interchange and two more directly with computational aspects of mathematical programming.

Of the data interchange papers one introduces a new library for electronic interchange of mixed integer programming problems. The existence of such libraries are of major importance when new algorithms are implemented and tested in practice.

The second paper discusses the possibility of using C++ classes as vehicles for easy data interchange in stead of the traditional MPS format files.

Linear programming and interior point methods is a very active research field, and the third paper presents a dual affine scaling method particularly well suited for problems with many free variables. Computational results are reported both on netlib problems and on a test set originating in an engineering application.

Tabu search and simulated annealing are two paradigms for constructing local search heuristics for combinatorial optimization problems. It seems difficult to find a fair way of comparing these. In the fourth paper, a method based on the number of neighbour evaluations is proposed, and experimental results are reported for the case of simulated annealing.

The next issue of the Bulletin will be published in December 1993.

Jens Clausen

Mathematical Programming Society

Committee on Algorithms Bulletin

NO. 22
August 1993

JENS CLAUSEN
FAIZ A. AL-KHAYYAL
EDITORS