DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 00, 1990

ABSTRACT. This paper discusses a new algorithm for solving the assignment problem. This algorithm, called the Approximate Dual Projective algorithm for assignment (ADP/A), is a variant of the Karmarkar interior point algorithm for LPs, specialized to solve the assignment problem. Computational results are reported on various classes of assignment problems. These results indicate that this method, holds promise for solving large assignment problems.

1. Introduction

In 1984, Karmarkar proposed a new polynomial time algorithm for solving linear programming problems [**11**]. This algorithm, the Karmarkar projective algorithm, and many of its variants are known as interior point methods. These methods have been studied and found to be effective solution techniques for solving large general LP problems [**1, 13, 16**]. The purpose of the study presented here is to evaluate the performance of one variant of the interior point method for solving a simple class of LPs, namely, the assignment problem. The variant we study is called the approximate dual projective (ADP) method. The ADP method computes good discrete approximations to the continuous trajectory of the Karmarkar projective method. Asymptotically, ADP trajectory approaches the Karmarkar projective trajectory. We have implemented the ADP method using the preconditioned Conjugate Gradient(CG) technique to solve the linear system that arises in computing the improving direction. Our research software that implements the ADP method has been found to be effective in solving large general LPs arising in many diverse areas of engineering [**13**].

We have adapted the ADP method to solve large assignment problems. This adapted algorithm, the Approximate Dual Projective algorithm for assignment problems (to be referred to as ADP/A hereafter) is identical to our general purpose ADP method except for the stopping criterion; we simply terminate the dual iterates by identifying the optimal assignment . This identification of

1991 *Mathematics Subject Classification* Primary .
Key words and phrases. .

optimal assignment is guided by the current dual slacks and the complementary slackness. Fig. 1 shows the overall structure of the algorithm. The algorithm

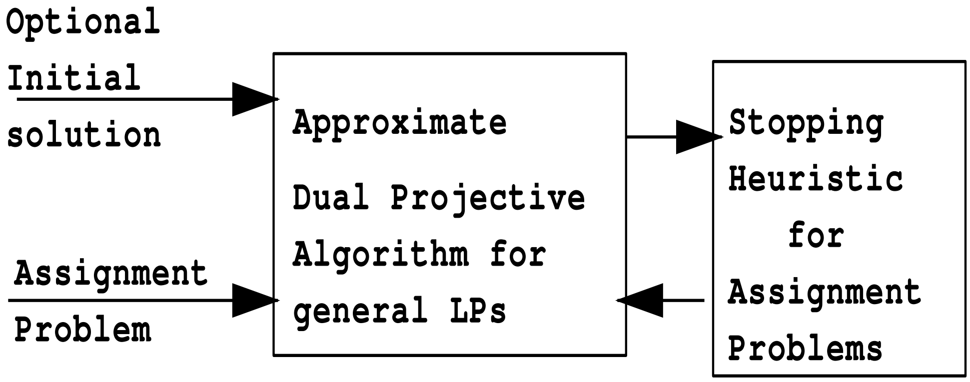

FIGURE 1. Structure of ADP/A Algorithm

inside the first box is our ADP method. We give as input to the ADP method, the assignment problem formulated as an LP, and optionally an initial dual solution. After the ADP method has made sufficient progress towards optimality, we begin checking, at a set frequency, whether we can identify the optimal assignment from the current solution. If this heuristic succeeds, we terminate the ADP method; otherwise the ADP method continues with the LP solution. It should be noted that the ADP method has no knowledge that it is solving an assignment problem; in particular, no attempt is made to perform integer arithmetic. There are only two modules that exploit the special structure of the assignment problem, the initial solution evaluation module and the stopping heuristic. These two modules are outside the main ADP algorithm. This is an example of how interior point methods can exploit the special nature of the problem without sacrificing the ability to solve general LPs. Applications of assignment problems which involve side constraints can be directly solved by our algorithm, unlike specialized combinatorial algorithms for assignment problems.

We report computational results for randomly generated dense assignment problems having as many as 30 thousand nodes and 900 million arcs. We also report on the 4 classes of DIMACS problem generators. We compare our results with a version of the auction algorithm. Section 2 gives an overview of the ADP method. Section 3 discusses the stopping heuristic. Section 4 contains the computational results, followed by a discussion of the results in Section 5. Finally, Section 6 discusses the conclusions of this study.

2. An Overview of the Approximate Dual Projective Algorithm

In this section, we give a brief overview of the Approximate Dual Projective (ADP) algorithm for linear programming. For a complete description of the algorithm please see [**13**]. As the name implies, ADP method is an approximation to the projective algorithm of Karmarkar [**11**] applied to the dual. The projective algorithm has very nice stability properties, since the iterates of the projective algorithm always stay on the central trajectory [**3**]. The ADP method makes discrete approximations to the continuous central trajectory. Asymptotically, after a large number of iterations, the discrete approximations converge to the central trajectory. The discrete approximation consists of two steps; the objective step, and the reciprocal-estimates-improvement (REI) step. Figure 2 depicts the ADP method. In the figure, the optimal vertex is shown as a solid

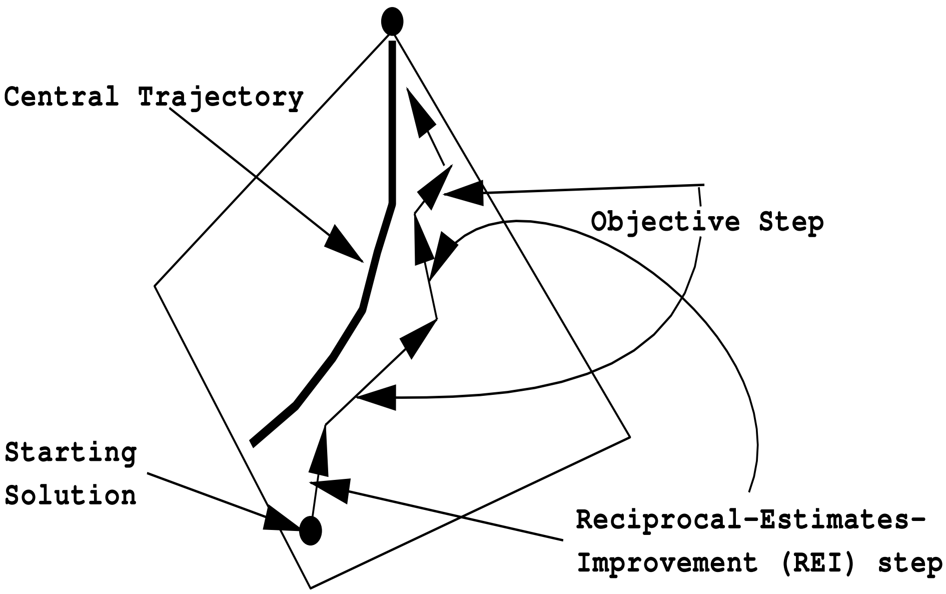

FIGURE 2. Pictorial View of the Approximate Dual Projective(ADP) algorithm

bullet at the top, and the central trajectory is shown as a double solid line. The objective step is the dual affine scaling step [**1, 13**] that improves the objective function. The REI step has the effect of bringing the iterate closer to the central trajectory. The name reciprocal estimates improvement comes from the fact that on the central trajectory of the dual polytope, the reciprocals of slack variables are exact primal feasible solutions upto a multiple [**13**]. Thus if we take a step in the dual polytope that improves the primal feasibility of the reciprocals of slacks, then this necessarily implies that the step brings the iterate closer to the central trajectory. The approach of improving reciprocal estimates is an interesting interpretation of the traditional way of centering, i.e., that of minimizing the potential function $-\sum \ln s_i$, where s_i is the dual slack variable. In fact, the linear systems solved by the newton-method-based minimization of the potential function and the REI step are identical. The ADP/A algorithm is similar to the BLCA algorithm of Todd [**18**], the affine scaling algorithms of Gonzaga [**9**],Barnes, Chopra, and Jensen [**2**], and many others. All these algorithms use affine scaling with centering on the primal problem. Recently, dual affine scaling algorithm with centering was studied by Hipolito [**10**].

Generally the ADP method consists of two phases. In Phase 1, the dual phase, the dual of the LP problem is solved; and in phase 2, a primal solution is obtained, starting with the estimates from the converged dual solution. At the end of execution of both phases, a primal-dual pair with a small relative duality gap is obtained. For the ADP/A algorithm for solving the assignment problems, since we terminate the dual iterates by identifying the optimal assignment, only Phase 1 of the ADP method is executed.

We now give a brief overview of the ADP method. Consider the LP problem, with upper bounds in standard primal form:

$$\min \; c^T x \tag{2.1}$$

subject to

$$Ax = b \tag{2.2}$$

$$0 \leq x \leq u \; . \tag{2.3}$$

Here $A \in \mathcal{R}^{m \times n}$, $c, x, u \in \mathcal{R}^n$, and $b \in \mathcal{R}^m$. Introducing dual variables y and z, corresponding to (2.2) and (2.3) respectively, the dual of (2.1)–(2.3) becomes

$$\max \; b^T y + u^T z \tag{2.4}$$

subject to

$$A^T y + z \leq c \tag{2.5}$$

$$z \leq 0 \; . \tag{2.6}$$

Defining

$$s_2 \triangleq -z \tag{2.7}$$

and introducing slack variables s_1 in (2.5), the dual (2.4)–(2.6) becomes

$$\max \; b^T y - u^T s_2 \tag{2.8}$$

subject to

$$A^T y - s_2 + s_1 = c \tag{2.9}$$

$$s_1, s_2 \geq 0 \; . \tag{2.10}$$

We can now develop a variant of the dual projective algorithm for solving (2.8)–(2.10). The scaling matrix D is defined in partitioned form as

$$D \triangleq \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} ; \tag{2.11}$$

where

$$D_1^{-1} \triangleq \text{diag}\ (s_1) \tag{2.12}$$

and

$$D_2^{-1} \triangleq \text{diag}\ (s_2)\ . \tag{2.13}$$

Let Δy, Δs_1, and Δs_2 be the ascent directions used in the algorithm. Given below are the formulas for Δy, Δs_1, and Δs_2. The derivation of these formulas is identical to the derivation in [**1**] except for the treatment of upper bounds on the primal variables. The derivation of these formulas in the presence of upper bounds can be found in [**12**].

$$\Delta y = (A\tilde{D}^2A^T)^{-1}(b - AD_1^2D_e^2u) \tag{2.14}$$
$$\Delta s_1 = -D_e^2D_2^2A^T(A\tilde{D}^2A^T)^{-1}(b - AD_1^2D_e^2u) - D_e^2u \tag{2.15}$$
$$\Delta s_2 = D_e^2D_1^2A^T(A\tilde{D}^2A^T)^{-1}(b - AD_1^2D_e^2u) - D_e^2u \tag{2.16}$$

where

$$D_e^{-2} = D_1^2 + D_2^2 \tag{2.17}$$
$$\tilde{D}^2 = D_1^2 - D_1^2D_e^2D_1^2\ . \tag{2.18}$$

The ADP algorithm is given below.

ALGORITHM A: APPROXIMATE DUAL PROJECTIVE ALGORITHM: DUAL PHASE.

(1) Given an initial interior starting point y^0, s_1^0, s_2^0 and parameters α, and ϵ, set the iteration counter $i = 0$; switch_criterion = false.

(2) **WHILE** switch_criterion is false **DO**

  2.1 // Objective function improving step.

   a) Compute ascent directions Δy, Δs_1, and Δs_2 using equations (2.11) to (2.18).

   b) // compute maximum step length permissible to keep s_1 and // s_2 nonnegative.
   $\beta = \min\left(\min_{\Delta s_{1k}<0} -\frac{s_{1k}^i}{\Delta s_{1k}},\ \min_{\Delta s_{2m}<0} -\frac{s_{2m}^i}{\Delta s_{2m}}\right)$

   c) // Compute a fraction of the maximum step length.
   $\beta = \alpha\beta$

d) // Update the solution.
$y^{i+1} = y^i + \beta \Delta y$
$s_1^{i+1} = s_1^i + \beta \Delta s_1$
$s_2^{i+1} = s_2^i + \beta \Delta s_2$
$i = i + 1$

2.2 // Check if relative improvement in objective function is less than ϵ
IF $\left((b^T y^i - u^T s_2^i) - (b^T y^{i-1} - u^T s_2^{i-1}) < \epsilon |(b^T y^{i-1} - u^T s_2^{i-1})|\right)$
THEN
switch_criterion = true
END_IF

2.3 // Reciprocal estimates step.
a) Apply the reciprocal estimates to the current iterate and obtain direction vectors Δy, Δs_1, and Δs_2.
b) reexecute steps 2.1.b to 2.1.d to update the solution.

END_WHILE

Several observations about ALGORITHM A are worth making.

- The reciprocal estimates applied in step 2.3.a is based on an elegant mathematical relationship between the dual slacks and the primal variables of the LP. The relationship is that, on the central trajectory [**3**], the reciprocals of dual slacks are primal feasible, up to a scalar multiple. This property is exploited in ADP to perform approximate centering using the reciprocal estimates improvement (REI) algorithm. The details of this algorithm can be found in [**13**].
- The termination criterion in step 2.2 is the normal termination criterion. This step checks whether the relative improvement in the objective function value is less than ϵ. For the ADP/A method, we have to replace this step by the following step.
  (a) call ALGORITHM ASSIGN, after sufficient progress is made on the dual iterates.
  (b) if ALGORITHM ASSIGN succeeds, stop; else check relative improvement in objective function value (identical to step 2.2 in ALGORTIHM A).

  In all our computational experiments ALGORITHM ASSIGN always succeeded in finding the optimal assignment. Of course, the number of dual iterations that have to be performed before ALGORITHM ASSIGN succeeds depends on the nature of the assignment problem; sometimes we had to perform many iterations of the ADP method before ALGORITHM ASSIGN succeeded while at other instances the algorithm succeeded on its first try (after 10 dual iterations).
- The computationally intensive step is the solution of the positive definite linear system of equations encountered in evaluating the ascent direc-

tion. For this step, we have used the preconditioned conjugate gradient method, with a dynamically adaptive preconditioner.

3. Adapting ADP Algorithm for solving the Assignment Problem

We now discuss the ADP/A method, the adapted ADP method to solve the assignment problems. Basically, two items of adaptation are incorporated in ADP/A method; the stopping heuristic and the initial solution. We discuss both of these topics below.

It is well-known that the assignment problem can be formulated as a linear programming problem [**17**]. In a symmetric assignment problem, we are given n source nodes and n destination nodes. A source node i maybe connected to a destination node j by an edge with cost c_{ij}. Let E be the set of all edges in the problem.

The primal problem can be easily formulated as follows. We associate a 0-1 variable x_{ij} with each edge. Since at least one edge emanating from a source has to be selected, we get the constraint

$$\sum_{j,(i,j)\in E} x_{ij} = 1$$

We get similar constraints for the destination node.

$$\sum_{i,(i,j)\in E} x_{ij} = 1$$

In addition x_{ij} is constrained to be between 0 and 1.

$$0 \leq x_{ij} \leq 1$$

If we need to assign nodes such that the selected edges have minimum cost then the objective of the linear program would be to minimize $\sum_{i,j} c_{ij} x_{ij}$ and the primal form of the linear program can be written in the form

$$\min c^T x$$

$$\text{s.t. } Ax = b$$

$$0 \leq x \leq 1$$

If we ignore the upper bound constraints, which are in any case redundant then the dual obtained is of the form,

$$\min b^T y$$

$$\text{s.t. } A^T y \leq c$$

Here, the dual variables correspond to source and destination nodes and the constraints correspond to the edges in the problem. So if z_i is a variable corresponding to source node i and w_j is a variable corresponding to destination node j then if there exists an edge between these nodes of cost c_{ij} then the dual constraint is of the form

$$z_i + w_j \leq c_{ij} \tag{3.1}$$

3.1. Initial Solution. For general LPs, ADP method starts with an initial feasible solution by introducing artificial variables, if necessary. In ADP/A method, since the dual constraints are simple, they obviate the need for artificial variables. In ADP/A method, we start from an interior feasible point which is obtained as follows. Since the dual constraints are of the form

$$z_i + w_j + s_{ij} = c_{ij} \tag{3.2}$$

First, we determine

$$\alpha = \min c_{ij}$$

and then set for all i, j and some $\beta > 0$

$$z_i = w_j = (\alpha \ - \ \beta)/2$$

$$s_{ij} = c_{ij} - \alpha + \beta$$

It may be easily verified that this is a feasible starting point for the ADP algorithm. For our computational experiments, β was set at .1 times the maximum of the absolute value of costs. This usually resulted in the ratio of maximum to minimum s_{ij} to be about 10.

3.2. Identification of Optimal Assignment from Dual Solution (Stopping Heuristic). As mentioned earlier, we can identify the optimal assignment and prove its optimality from the state of the dual variables by using the complementary slackness principle. Hence there is no need to switch to the primal phase for finding the optimal assignment. The method for identifying the optimal assignment from the dual variables can be described as follows. Assign each source node i to a sink node j, i.e., select edge (i,j), such that

- j is not yet assigned to any source node
- s_{ij} is minimum slack among all sink nodes satisfying above condition.

If the above scheme fails to assign every source node then we quit the stopping heuristic. On the other hand, if we get a feasible assignment with the above scheme then we check the assignment for optimality. For optimality checking, we define temporary dual variables $(\tilde{z}, \tilde{w})$ as follows. For each edge (i,j) selected above, set $\tilde{z}_i = z_i + s_{ij}/2$, and $\tilde{w}_j = w_j + s_{ij}/2$ (note that if we have a feasible assignment then all the temporary dual variables $(\tilde{z}, \tilde{w})$ will be set). We then check whether $\tilde{z}$ and $\tilde{w}$ are feasible solutions of the dual. If they are feasible then we have found the optimal solution and we have also proved optimality (by enforcement of complementary slackness) The optimal dual solution is $(\tilde{z}, \tilde{w})$ and the optimal primal solution is $\tilde{x}$ where $\tilde{x}_{ij}$ is 1 if the edge (i,j) was selected above and 0 otherwise. If $(\tilde{z}, \tilde{w})$ is not feasible, the assignment is declared not optimal and we continue with our ADP algorithm using (z, w) as the current iterate. It can be shown that eventually, the dual phase will converge to a solution from which we can identify the optimal assignment using the scheme described above.

The stopping heuristic discussed above assumes a symmetric *feasible* assignment problem. The heuristic is guaranteed to find the optimal assignment, if it is unique. This is because the slack variables corresponding to the edges in the optimal assignment converge to zero as the dual iterates converge to the optimal solution. When there are multiple optimal assignments, i.e., a facet of the primal polytope is optimal, the ADP/A algorithm goes to the center of the facet. In this case, the heuristic may fail to find an optimal assignment. In all our experiments, we have perturbed the objective vector randomly in the last few bits, to guarantee a unique optimal solution (since ADP/A works with double precision cost coefficients, it is easy to perturb the mantissa of a double precision variable). Of course, the final optimality check is done on the original objective vector.
ALGORITHM ASSIGN shows the pseudocode for the stopping heuristic.

ALGORITHM ASSIGN:.

Input:

(1) m, the number of nodes in each partition of the bipartite graph.
(2) E, the list of edges in the bipartite graph.
(3) Current values of dual variables z_i, w_j, and s_{ij}.

Output: return a value of 1 if optimal assignment was found along with the dest array containing the optimal assignment; 0 otherwise

(1) Initialize: $source(k) \leftarrow 0; 1 \leq k \leq m$
(2) **FOR** $1 \leq i \leq m$; **DO**
  2.1 $\alpha \leftarrow \infty; dest(i) \leftarrow 0$
  2.2 **FOR** j such that $(i, j) \in E$ *and* $source(j) = 0$ **DO**
      IF $s_{ij} < \alpha$ **THEN** $\alpha \leftarrow s_{ij}; dest(i) \leftarrow j$
      END_IF
    END_FOR
  2.3 **IF** $(dest(i) = 0)$**THEN** return(0);
    ELSE
      $j \leftarrow dest(i)$
      $source(j) \leftarrow i$
      $\tilde{z}_i \leftarrow z_i + s_{ij}/2$
      $\tilde{w}_j \leftarrow w_j + s_{ij}/2$
    END_IF
  END_FOR
(3) **IF** $(\tilde{z}, \tilde{w})$ is feasible for the dual **THEN** return(1,dest);
  ELSE return(0)
  END_IF

4. Computational Results

We now describe the computational results of the ADP/A method. ADP/A algorithm always takes as input an LP posed in standard primal form; i.e., min $c^T x$ subject to $Ax = b$, $x \geq 0$. By changing the sign on the objective vector coefficients, we converted the maximum assignment problem returned by the generator to an equivalent minimum assignment problem. Ofcourse the optimal solution of the minimum assignment problem is identical to the optimal solution of the maximum assignment problem. Also, the optimal objective value of the maximum assignment problem is equal to the negative of the optimal objective value of the minimum assignment problem.

Our implementation of ADP/A uses a preconditioned conjugate gradient algorithm to solve the linear system at each iteration. The preconditioned conjugate gradient method has a set of parameters which determine its behaviour. Parameters for CG and the range over which the values were varied for the different problem classes are given below in table 1.

parameter type	range of values
step size parameter (α in ALGORITHM A)	.9 to .99
$cos\theta$ stopping criterion for CG	10^{-6} to 10^{-8}
Residue stopping criterion for CG	10^{-5} to 10^{-7}
Iteration limit for CG	10 to 100

TABLE 1. Parameters of the ADP/A Algorithm that were Used for Different Problem Classes

To evaluate the performance of ADP/A we compare the results with the latest version of the Forward/Reverse Auction code [**4**, **6**, **5**] obtained from Prof. D. Castañon. Originally, we ran a version of auction code obtained from Prof. Bertsekas, but found that this code had integer overflows because of initial scaling of costs and a large cost range. When we communicated this problem to Prof. Castañon during the DIMACS conference, he supplied us with a modified auction code that uses double precision locations to store dual variables. Another feature of this auction code is the *absence of initial cost scaling*. This new auction code is what we have used in our comparisons. This code is written in fortran and was compiled using the `f77` compiler, with the optimization flag set to -O2.

The parameters that are hardcoded in the program are as follows (all the parameters are defined as real and N is the number of nodes in the bipartite graph):

- BEGEPS = 1.0 / (N + 1)
- ENDEPS = 1.0 / (N + 1)
- STARTINCR = 1.0 / (N + 1)
- FACTOR = 10.0

All runs were carried out on a *Silicon Graphics Machine*, having a Risc-based architecture with 25MHZ clock. The SILICON GRAPHICS(R) uses a variant of the UNIX[1] Operating System, called the IRIX(R) Operating System. The ADP/A algorithm is implemented in FORTRAN and C. In our experiments, the code was compiled using the `f77` and `cc` compilers with optimization flag `-O2`. All times reported are user times given by the system call `times()`. First, a call to `times()` is made just before the call to the actual solver. This gives us the user cpu time consumed by the program so far. After the call to the solver returns, another call to `times()` is made to obtain the user cpu time. The difference between the two times is reported as the cpu time consumed by the solver. Thus, all problem generation and problem read times are **excluded** from the reported times. This remark applies to both the ADP/A and auction experiments.

To benchmark the cpu speed of the SGI machine, the benchmark experiments from pub/netflow/benchmarks were run on the machine. Since the ADP/A implementation uses both C and F77, both sets of benchmarks were run. Tables 2 and 3 show the results of the benchmark experiments. The mean and standard

	Test 1		Test 2	
	Mean	Std. Dev.	Mean	Std. Dev.
real time	2.0s	.03s	22.9s	2.2s
user time	1.9s	.03s	22.4s	2.0s
sys time	.05s	.00s	0.3s	.01s

Table 2. Cpu times for Benchmarks run on the SGI machine–C benchmarks

	Test 1		Test 2	
	Mean	Std. Dev.	Mean	Std. Dev.
real time	1.8s	.06s	3.2s	.35s
user time	1.6s	.13s	2.9s	.42s
sys time	.16s	.00s	.23s	.00s

Table 3. Cpu times for Benchmarks run on the SGI machine–f77 benchmarks

deviation of times are the results of 4 experiments varying the optimization level of the compilers. (-O1 to -O4). The fortran benchmarks consist of running the NETFLO algorithm [**14**]on two minimum cost network flow problems generated by NETGEN [**15**].

There are five assignment problem classes we have studied. The problem classes differ in the cost range and sparsity of the bipartite graphs. The five problem classes are given below.

[1]UNIX is a registered trademarks of AT&T.

(1) Assign-Geometric(AGM):In this class, points are distributed on a plane and divided into two groups; red and blue. Edges connect red and blue points with edge costs equal to the Euclidean distance between the points.
(2) Assign-Fixed-cost(AFX): Problems in this class have a fixed degree of $N/16$ and the edge cost between nodes i and j is set to $100 * i * j$.
(3) Assign-High-Cost(AHC): Problems in this class have a fixed degree of $2\ log_2 N$ and the edge costs are uniformly distributed between $[0,10^8]$.
(4) Assign-Low-Cost(ALC): Problems in this class are identical to the problems in AHC, except for the costs being uniformly distributed between [0,100].
(5) Completely dense graphs(CDG): These problems are completely dense with the costs uniformly distributed between [0,1000] or [0-10000].

The first four problem classes are from the DIMACS problem set. For the problems from these four classes, Appendix A gives the seeds we used to generate the problems along with the optimal objective function value for the problem instances. The fifth problem class (CDG) is generated by our generator. This problem class is similar to the Assign-Xhi-degree problem set in DIMACS. Thus we did not run the Assign-Xhi-degree problem class. Also, Assign-two-cost problem class from DIMACS proved to be too trivial for both ADP/A and auction algorithms. Therefore we do not report on this problem class in this paper.

Now we will describe the computational results. Table 4 summarizes the results for the AGM problem class. The first column indicates the problem size in

Nodes	ADP/A		AUCTION		SPEEDUP $\frac{\text{AUCTION}}{\text{ADP/A}}$
	Mean	Std. Dev.	Mean	Std. Dev.	
128	12s	.5s	7m24s	4m01s	37
256	47s	1s	1h11m05s	7m40s	78
512	3m34s	42s	39h40m51s	-	667
1024	21m56s	4m48s	**	**	**

** Auction was not run because of projected cpu time requirements

TABLE 4. Performance comparison with Auction on Geometric Cost Assignment Problems(AGM)

terms of the number of nodes in *each partition*. The mean and standard deviation of cpu times are shown in the table, along with the speedup. The mean and standard deviation are computed using a sample size of three runs with different seeds for the random numbers (See Appendix A). The speedup is the ratio of the means of auction running time to ADP/A running time. For this class of problems ADP/A is seen to be one to two orders of magnitude faster than the version of the auction code obtained from Prof. Castañon. It is somewhat puzzling why auction takes excessive cpu time for this class of problems compared to the other classes. There are several possible explanations. It is possible that the auction code that we obtained from Prof. Castañon implements the UFR (Unscaled Forward Reverse [**8**]) algorithm. Prof. Castañon in [**8**] states that UFR is particularly inefficient for the AGM class of problems. Another possible explanation for long running times of auction is the existence of prolonged price wars in the instances we are running. In [**8**] it is stated that the problem symmetry of this class produces price wars both in forward and reverse directions.

Table 5 summarizes the results for the AFX class of problems. Problems in

Nodes	ADP/A		AUCTION		SPEEDUP $\frac{\text{AUCTION}}{\text{ADP/A}}$
	Mean	Std. Dev.	Mean	Std. Dev.	
128	3s	.3s	.2s	.1s	.1
256	11s	.1s	4s	1s	.4
512	46s	1s	1m49s	1m03s	2
1024	4m36s	9s	1h31m50s	1h38m54s	20

TABLE 5. Performance comparison with Auction on Fixed Cost Assignment Problems(AFX)

this problem class have a degree of N/16, where N is the number of nodes in the graph. Thus these problems are moderately dense. ADP/A is again seen to perform one order of magnitude better than auction. Interestingly, ADP/A starts performing better than auction after 512 nodes, whereas for AGM class the ADP/A performs better even on smaller problems. One explanation is that problems in AGM class are completely dense. The interior point code is able to quickly isolate interesting edges from this dense problem. Since AFX class is only moderately dense, this nice property of the interior point method takes effect on bigger problems. This phenomenon also happens in CDG class, as discussed below.

Now we describe the CDG class of problems. As mentioned before, this is our own generator for dense assignment problems. To be able to solve very large dense problems, we need to perform some *sifting* during the generation phase. We give as input to the random instance generator the number of source and destination nodes m, the range of the costs (L, U) and an additional parameter called degree d which we describe below. Though we generate dense assignment problems, we drop many unviable edges, in the generation phase. The degree parameter is used to specify how many edges per node have to be retained on an average. Good estimates for degree, such that the resulting reduced graph has a high probability of containing the optimal assignment of the original dense graph, can be derived from the theory of random graphs [7]. The sifting gives us a reduced linear programming problem which can fit into the virtual memory limits of our machine. However, on obtaining the optimal solution, we regenerate all the edges and verify the optimality of the solution for the entire dense assignment problem. A similar approach of dropping low cost edges and later verifying optimality of solution for the entire dense problem was adopted for the auction algorithm. For both algorithms, the generation time is excluded from the running times reported.

Table 6 summarizes the results obtained in these experiments. We report

Nodes	Cost Distributions					
	0 - 1000			0-10000		
	ADP/A	AUCTION	SPEEDUP $\frac{\text{AUCTION}}{\text{ADP/A}}$	ADP/A	AUCTION	SPEEDUP $\frac{\text{AUCTION}}{\text{ADP/A}}$
1000	19s	7s	0.36	21s	8s	0.38
2000	1m14s	26s	0.35	1m30s	29s	0.32
3000	2m26s	2m16s	0.93	3m07s	2m42s	0.86
5000	5m17s	8m17s	1.56	6m05s	7m54s	1.30
7500	12m06s	20m15s	1.67	12m13s	1h39m20s	8.13
10000	18m36s	49m08s	2.64	19m13s	5h07m57s	16.02
20000	1h27m37s	1h58m03s	1.34	1h25m34s	8h38m23s	6.05
30000	3h17m02s	12h04m08s	3.67	3h19m10s	140h50h49s	42.43

TABLE 6. Performance comparison with Auction on Dense Assignment Problems

here the timings for two different cost distributions - one in which the costs vary from 0 to 1000 and the second in which the cost distribution is from 0-10000. It should be noted that for both the algorithms, the times include the time spent in verifying the optimality of the solution for the entire dense assignment problem.

The results seem to indicate the following.

- ADP/A is slower than auction on smaller instances. This is again the same phenomenon we observed in AFX problem class. For moderately dense problems, auction is faster on small size problems. However for larger instances ADP/A seems to be faster than auction with a speedup of more than 40 times in one instance of the thirty thousand node assignment problem. Observe that even though we are generating completely dense problems, we sift out many edges resulting in a very sparse problem. In fact the average degree for all the problems upto 5000 nodes was less than 30. The key word here is the *average* degree. There were some minor variations between degrees of various nodes. This is different from the *fixed degree* sparse graphs of the AHC and ALC problem classes discussed below. In these latter problem classes, every source node has the same *fixed* degree. Thus the graphs are somewhat regular. As discussed below, auction performs much better than ADP/A on such graphs.

- The change in ADP/A timings with increase in cost range is almost negligible.

Tables 7 and 8 summarize the computational results for the AHC and ALC problem classes.. These two problem classes consist of problems with a fixed

Nodes	ADP/A		AUCTION		SPEEDUP $\frac{\text{ADP/A}}{\text{AUCTION}}$
	Mean	Std. Dev.	Mean	Std. Dev.	
1024	17s	.6s	.4s	.1s	43
2048	36s	2s	2s	.4s	18
4096	2m12s	4s	8s	4s	17
8192	3m22s	19s	10s	4s	20
16384	9m05s	29s	32s	10s	17
32768	24m23s	2m19s	51s	4s	29

TABLE 7. Performance comparison with Auction on Assign-High-Cost(AHC) Assignment Problems

Nodes	ADP/A		AUCTION		SPEEDUP $\frac{\text{ADP/A}}{\text{AUCTION}}$
	Mean	Std. Dev.	Mean	Std. Dev.	
1024	15s	2s	.3s	.1s	50
2048	39s	1s	1s	.2s	39
4096	1m58s	2s	3s	.2s	39
8192	5m01s	9s	8s	.5s	38
16384	13m23s	38s	40s	17s	20
32768	41m04s	2m19s	2m25s	1m26s	17

TABLE 8. Performance comparison with Auction on Assign-Low-Cost(ALC) Assignment Problems

degree of $2log_2N$. Thus these graphs are somewhat regular and sparse. For these classes of problems auction does extremely well. It performs an order of magnitude faster than ADP/A. It appears that the overhead involved in isolating important dual slacks in ADP/A is too excessive and cannot be justified for sparse problems since the number of edges being eliminated is too small.

5. Discussion

The computational results discussed above demonstrate some interesting behavior of the ADP/A algorithm. Firstly, we notice that the coefficient of variation of cpu times (standard deviation / mean) for ADP/A is small. Among all the results presented above, the largest coefficient of variation is .2. This indicates that ADP/A has little sensitivity to the problem instances.

Secondly, the cpu time of ADP/A grows almost linearly with the number of edges in the graph for each of the five problem classes we have studied. For instance, for the Assign-Fixed-Cost problem class (Table 5) the mean cpu time increases by a factor of 4 as you double the number of nodes; i.e., as you increase the number of edges by a factor of 4. Moreover, cpu time taken by ADP/A for a problem instance is roughly proportional to the number of edges regardless of the problem structure. For example, consider the 1024 node problems in different problem classes. The cpu time for Assign-Geometric is 21m56s (Table 4). The number of edges in this problem is about 1 million. In Assign-Fixed-Cost the cpu time is 4m36s (Table 5) and the number of edges is about 1/8 of a million. Also, the cpu time for Assign-High-Cost is 17s (Table 7) and the number of edges is about 20,000. The ratios of cpu times correspond roughly to the ratios of number of edges. This demonstrates uniformity in the behaviour of the ADP/A algorithm over the problem classes studied here. Not only does it show little dependence on the range of costs but also little dependence on the problem structure.

Thirdly, another point we notice is that ADP/A is able to solve very large problems without degradation in performance . Thus we expect ADP/A to asymptotically outperform other algorithms; e.g., for the Assign-Geometric problem class, the speedup of ADP/A increases with problem size (Table 4) and for the Assign-Low-Cost problem class the speedup of auction over ADP/A steadily decreases with increase in problem size (Table 8).

6. Conclusions

We have applied a general purpose approximate dual projective variant of the interior point method to solve large assignment problems. We have made minor adaptations in the code that help early identification of the optimal assignment from the dual iterate.

This general purpose approach for solving assignment problems has some benefits:

- ADP/A method can be directly applied to an assignment problem with side constraints. We simply add these additional constraints to the LP and invoke ADP/A. No special algorithmic modification of ADP/A is required.
- ADP/A method can be directly applied to assignment problems with real (nonintegral) costs since ADP/A makes no assumptions about costs being integers.
- The ADP/A method can be easily extended to assignment problems with convex costs. This is because of the general purpose nature of interior point method.

This study has illustrated how one can exploit the special nature of an LP problem without sacrificing the generality of the algorithm. We are currently conducting research on applying the Approximate Dual projective method to solve network flow and multicommodity network flow problems.

7. Appendix A: Problem seeds and Optimal Objective Values

This appendix lists the problem instances we have generated for our experiments. The four problem classes discussed below are from the DIMACS problem set and are in the public domain. For each problem, we have given the random seed used and the optimal objective function value obtained.

7.1. Assign-Geometric Class of Problems. These problems are generated using the dcube.c generator in the DIMACS library. The parameters used to generate the problems are given below.
Nodes (N)
dimension 2
maxloc 1000000
seed (X)
The various sizes of problems generated, the corresponding seeds and the optimal objective function values are listed in table 9

Nodes (N/2)	seed (X)	Optimal Objective Value
128	270001	98590148
128	270002	95517946
128	270003	93248933
256	270001	198237217
256	270002	192965978
256	270003	194844097
512	270001	391309422
512	270002	391237799
512	270003	391148927
1024	270001	782311551
1024	270002	786202387
1024	270003	779219446

TABLE 9. Seeds and Optimal Objective Values for Assign-Geometric Problems

.

7.2. Assign-Fixed-Cost Class of Problems. This class of problems is generated by assign.c with the parameters given below.
Nodes (N)
sources (N/2)
degree (N/16)
maxcost 100

multiple
seed (X)
Table 10 lists the seeds and the optimal objective values.

Nodes (N/2)	seed (X)	Optimal Objective Value
128	270001	175997100
128	270002	175913300
128	270003	175884000
256	270001	1403859100
256	270002	1403897200
256	270003	1403886100
512	270001	11209411800
512	270002	11209513900
512	270003	11209025600
1024	270001	89580255700
1024	270002	89580094400
1024	270003	89579848400
2048	270001	716240798500
2048	270002	716240324300
2048	270003	716240995800

TABLE 10. Seeds and Optimal Objective Values for Assign-Fixed-Cost Problems

7.3. Assign-High-Cost Class of Problems. These problems are generated using assign.c generator, with the following parameters.
nodes (N)
sources (N/2)
degree $(2\log_2(N))$
maxcost 100000000
Table 11 depicts the seeds and optimal values.

7.4. Assign-Low-Cost Class of Problems. These problems are generated using assign.c generator, with the following parameters.
nodes (N)
sources (N/2)
degree $(2\log_2(N))$
maxcost 100
Table 12 depicts the seeds and optimal values.

Nodes (N/2)	seed (X)	Optimal Objective Value
1024	270001	94804185690
1024	270002	95326326096
1024	270003	94809125860
2048	270001	190763064590
2048	270002	191079950281
2048	270003	190967830450
4096	270001	384168581871
4096	270002	384756241966
4096	270003	384331263783
8192	270001	772414482375
8192	270002	773314580225
8192	270003	771886833795
16384	270001	1550433745445
16384	270002	1551128099757
16384	270003	1550586814821
32768	270001	3113261955763
32768	270002	3112386719653
32768	270003	3112811065694

Table 11. Seeds and Optimal Objective Values for Assign-High-Cost Problems

Nodes (N/2)	seed (X)	Optimal Objective Value
1024	270001	95307
1024	270002	95847
1024	270003	95302
2048	270001	191793
2048	270002	192104
2048	270003	191955
4096	270001	386201
4096	270002	386766
4096	270003	386380
8192	270001	776507
8192	270002	777368
8192	270003	775955
16384	270001	1558512
16384	270002	1559224
16384	270003	1558666
32768	270001	3129458
32768	270002	3128530
32768	270003	3128957

Table 12. Seeds and Optimal Objective Values for Assign-Low-Cost Problems

References

1. Ilan Adler, Narendra K. Karmarkar, Mauricio G.C. Resende, and Geraldo Veiga, *An Implementation of Karmarkar Algorithm for Linear Programming*, Mathematical Programming **44** (1989), 297–335.
2. E.R. Barnes, S. Chopra, and D.L. Jensen, *A Polynomial Time Version of the Affine Scaling Algorithm*, Tech. Report 88-101, Graduate School of Business Administration, New York University, Washington Square, N.Y. 1988., 1988.
3. D.A. Bayer and J.C. Lagarias, *The Nonlinear Geometry of Linear Programming I, Affine and Projective Scaling Trajectories*, Transactions of American Mathematical Society **314** (1989), 499–526.
4. D. P. Bertsekas, *The Auction Algorithm for Assignment and Other Network Flow Problems*, Interfaces **20** (1990), no. 4, 133–149.
5. D.P̃. Bertsekas, *Linear Network Optimization: Algorithms and Codes*, MIT Press, Cambridge, MA, 1991.
6. D. P. Bertsekas and D.A. Castañon, *The Auction Algorithm for Transportation Problems*, Annals of Operations Research **20** (1989), 67–96.
7. Béla Bollobás, *Random Graphs*, Academic Press, London, 1985.
8. D.A. Castañon, *Reverse Auction Algorithms for Assignment Problems*, Proceedings of the DIMACS Conference, New Brunswick, New Jersey, October 1991.
9. C.C. Gonzaga, *Polynomial Affine Algorithms for Linear Programming*, Mathematical Programming **49** (1990), 7–21.
10. A. Hipolito, *A Long-step Linear Programming Algorithm Based on Quasi-Newton Inverse Barrier Centering*, Tech. Report 91-12, Dept. of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, October 1991.
11. N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica **4** (1984), 373–395.
12. N. K. Karmarkar and K.G. Ramakrishnan, *Implementation and Computational Results of the Karmarkar Algorithm for Linear Programming, Using an Iterative Method for Computing Projections*, Tech. Report 11211-891011-10TM, AT&T Bell Laboratories, Murray Hill, NJ, October 1989.
13. ———, *Computational Results of an Interior Point Algorithm for Large Scale Linear Programming*, Mathematical Programming **52** (1991), 555–586.
14. J.L. Kennington and R.V. Helgason, *Algorithms for Network Programming*, Wiley-Interscience Series, New York, NY,, 1980.
15. D. Klingman, A. Napier, and J. Stutz, *NETGEN: A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems*, Management Science **20** (1974), 814–822.
16. I.J. Lustig, R.E. Marsten, and D.F. Shanno, *Computational experience with a primal-dual interior point method for linear programming*, Tech. report, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, 1989, Tech report no SOR89-17.
17. C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 07632, 1982.
18. M.J. Todd, *A Low Complexity Interior Point Algorithm for Linear Programming*, SIAM Journal of Optimization **2** (1992), 198–209.

K.G. Ramakrishnan, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA, e-mail : kgr@research.att.com

N.K. Karmarkar, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA, e-mail: karmar@research.att.com

A.P. Kamath, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA and, Computer Science Dept., Stanford University, Stanford, CA 94305, e-mail: kamath@cs.stanford.edu