Product Form of the Inverse Revisited*

Péter Tar[1] and István Maros[1]

1 **Department of Computer Science and Systems Technology**
University of Pannonia, Veszprém, Hungary
`tar@dcs.uni-pannon.hu`

Abstract

Using the simplex method (SM) is one of the most effective ways of solving large scale real life linear optimization problems. The efficiency of the solver is crucial. The SM is an iterative procedure, where each iteration is defined by a basis of the constraint set. In order to speed up iterations, proper basis handling procedures must be applied.

Two methodologies exist in the state-of-the-art literature, the product form of the inverse (PFI) and lower-upper triangular (LU) factorization. Nowadays the LU method is widely used because 120-150 iterations can be done without the need of refactorization while the PFI can make only about 30-60 iterations without reinversion in order to maintain acceptable numerical accuracy.

In this paper we revisit the PFI and present a new version that can make hundreds or sometimes even few thousands of iterations without losing accuracy. The novelty of our approach is in the processing of the non-triangular part of the basis, based on block-triangularization algorithms. The new PFI performs much better than those found in the literature. The results can shed new light on the usefulness of the PFI.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Linear optimization, Simplex method, Basis inverse, Product form of the inverse

Digital Object Identifier 10.4230/OASIcs.SCOR.2012.64

1 Introduction

Our research aims to revisit the usability and effectiveness of the product form of the inverse in the simplex method in the light of the technological and algorithmic developments of the past decades. In Section 2 we present a brief literature overview of the simplex method and highlight the issues that play a key role in our investigations. These areas are the basis handling procedures, the property of sparsity and the numerical issues of the solution algorithm. In Section 3 we present our novel approach with a detailed description of the processing of the non-triangular part of the basis. In Section 4 a computational study is given to validate our results. Section 5 contains the conclusions.

* This publication/research has been supported by TÁMOP-4.2.2/B-10/1-2010-0025 project.

2 Literature overview

The history of linear optimization started in the early 1950's. The problem was originated by Dantzig [2]. One of the possible formulations of the problem is the standard form

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x}, \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$; $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$.

The simplex method gives the global optimum of a linear optimization problem. It also has a modular structure. Thus, loosely connected parts of the algorithm can be investigated separately. The simplex method is an iterative algorithm. The state of a simplex iteration can be described by a basis ($\mathbf{B}$) of the linear equation system. In each iteration one variable leaves the basis while another one enters. The whole process is started from an initial basis and ends with an optimal one if it exists.

2.1 Use of the basis in the simplex method

The efficiency of a simplex implementation can be measured by solution times. The total solution time depends on the number of iterations and the time consumed by one iteration. In our approach we focused on the time taken by an iteration. In each iteration two types of operations are necessary to facilitate a basis change:

$$\boldsymbol{\alpha} = \mathbf{B^{-1}a}, \quad \text{(or solve } \mathbf{B}\boldsymbol{\alpha} = \mathbf{a} \text{ for } \boldsymbol{\alpha}\text{)}, \tag{1}$$
$$\boldsymbol{\alpha}^T = \mathbf{a^T B^{-1}}, \quad \text{(or solve } \mathbf{B}^T\boldsymbol{\alpha} = \mathbf{a} \text{ for } \boldsymbol{\alpha}\text{)}, \tag{2}$$

where $\mathbf{B}$ is a basis of matrix $\mathbf{A}$ and $\mathbf{a}$ is some vector appearing in the algorithm. The computational effort needed to carry out these computations is significant. This highlights that the implementation of these two computations has an important influence on the performance of the simplex method. In the literature two major methods exist to perform operations with the basis. One is called the product form of the inverse (PFI), which we investigate in our work, the other is the elimination form (EFI) or lower-upper triangular (LU) factorization.

The PFI was introduced in [3]. The idea of this approach is to store the inverse of the basis as the product of special matrices.

$$\begin{aligned} &\mathbf{B^{-1}} = \mathbf{E_m E_{m-1}} \dots \mathbf{E_1} \\ &\mathbf{B_i^{-1}} = \mathbf{E_i B_{i-1}^{-1}}, \text{ for } i = 1, \dots, m, \end{aligned}$$

where $\mathbf{B_0^{-1}} = \mathbf{I}$ and, obviously, $\mathbf{B_m^{-1}} = \mathbf{B^{-1}}$

The $\mathbf{E_i}$ matrices are called elementary transformation matrices (ETMs). They differ from the unit matrix in only one column. During a basis change, this form can be updated by computing a new ETM. If the new basis is $\bar{\mathbf{B}}$, then $\bar{\mathbf{B}}^{\mathbf{-1}} = \mathbf{EB^{-1}}$.

The basic simplex operations of (1) and (2) are performed recursively:

$$\boldsymbol{\alpha} = \mathbf{E}_m\mathbf{E}_{m-1}\dots\mathbf{E}_1\mathbf{a} \tag{3}$$
$$\boldsymbol{\alpha}^T = \mathbf{a}^T\mathbf{E}_m\mathbf{E}_{m-1}\dots\mathbf{E}_1 \tag{4}$$

(3) is called FTRAN (Forward TRANsformation) while (4) is referred to as BTRAN (Backward TRANsformation).

The LU factorization of the basis in linear programming is presented in [7]. The idea is based on the triangular decomposition of the basis $\mathbf{B} = \mathbf{LU}$, where $\mathbf{L}$ is lower and $\mathbf{U}$ is upper triangular matrix. With this approach the FTRAN and BTRAN operations are computed by forward and backward substitutions. Details are omitted in this paper. The aim of our research was to investigate the PFI. During simplex iterations both forms (PFI and LU) can be updated by special formulas, so inversion (or factorization) is not necessary in every iteration. The update operations are relatively cheap in terms of computational effort.

2.2 Sparsity

Large scale real-life linear optimization problems usually have the property of sparsity. This means that there are very few non-zeros relative to the total size of the coefficient matrix $\mathbf{A}$. Experience shows that the average number of non-zero elements in a column is no more than 10, irrespectively of the size of the problem. During computations a vector can be transformed in such a way that the number of non-zero elements increases. This phenomenon is known as *fill-in*.

Independently of the PFI or LU method, triangular reordering of the basis must be performed during inversion (or factorization). The benefit of triangularization comes from sparse computing. It generally means that during inversion (or factorization) of the basis the number of non-zeros in the resulting inverse (or the LU matrices) can be kept low. In our work we focus on large-scale problems, thus exploiting sparsity is a key issue.

2.3 Numerical issues

During simplex iterations numerical errors (rounding, truncation and cancellation) can occur due to the finite precision of the computing architectures. As a general rule, double precision is used. In course of simplex iterations many additive and multiplicative operations are performed. As the computation carries on, these errors can add up and reach such magnitudes that can ruin the results unless numerical errors are handled properly.

As the basis changes numerical values become less accurate. Thus the vectors (like reduced cost, solution, etc.) computed by FTRAN and BTRAN operations lose accuracy, too. It can be said that the accuracy of the representation of the basis determines the accuracy of the resulting vectors. As such, it has a significant effect on choosing the incoming and outgoing variables. The accuracy of the computations can be controlled by reinversion (or refactorization) of the basis. This operation involves substantial computational effort.

The simplex algorithm is considered numerically stable if the following properties are satisfied during the solution process:

- The obtained solution is optimal.
- The numerical value of the objective value is (not strictly) monotonically improving.
- After reinvesion (or refactorization) the basis remains valid. (Numerical problems can lead to a linearly dependent set of vectors as a basis.)

3 Revisiting the product form of the inverse

Nowadays, the PFI is meant to be less effective than the LU form even if some examples exist for which PFI is superior. The idea of revisiting the product form emerged during the development of a simplex solver. We wanted to know the capabilities of the PFI in the light of recent technological and algorithmic developments. The basic problem with the PFI is that reinversion is needed after 30-60 iterations to maintain acceptable numerical accuracy, while this can go up to 100-200 if the LU form is used [8]. There were some efforts to create good PFI, but numerical problems weren't solved [6, 5].

In our work we focused on building a PFI based method and implementation that can solve real-life problems without reinverting frequently. We have built a system that can solve problems (from the *netlib* collection) using reinversion frequencies (number of iterations without reinverting) up to 300-3000.

We have implemented and investigated triangularization methods available in the literature by studying their strengths and weaknesses and also proposed numerical issues to be considered during the FTRAN and BTRAN operations. We present our approach in the following sections.

3.1 The inversion process

During inversion elementary transformation matrices are generated from the columns of the actual basis. An ETM is the result of a pivoting process on the basis. Inversion in product form can be described as follows.

1. Start with a matrix with known inverse, the identity matrix ($\mathbf{I}$) is a simple choice because $\mathbf{I^{-1}} = \mathbf{I}$.
2. Replace the columns of $\mathbf{I}$ with the columns of the basis, so the updating formula can be used.

The ETM for the update formula in the i^{th} step can be computed as:

$$\mathbf{E} = [\mathbf{e_1}, \dots, \mathbf{e_{i-1}}, \boldsymbol{\eta}_\mathbf{i}, \mathbf{e_{i+1}}, \dots, \mathbf{e_m}]$$

using

$$\begin{aligned}\boldsymbol{\eta}_\mathbf{i} &= \left[-\frac{v_i^1}{v_i^p}, \dots, -\frac{v_i^{p-1}}{v_i^p}, \frac{1}{v_i^p}, -\frac{v_i^{p+1}}{v_i^p}, \dots, -\frac{v_i^m}{v_i^p} \right] \\ \mathbf{v_i} &= \mathbf{B_{i-1}^{-1}} \mathbf{b_i},\end{aligned}$$

where $\mathbf{b_i}$ is column i of $\mathbf{B}$. In practical implementations, after a new ETM is formed, all the remaining columns are updated by it. In this way, the $\mathbf{v_i}$ vectors are available for the next step, there is no need for further computations. While the inverse of a matrix is unique the product form is not. This flexibility can be utilized to achieve some desirable goals. Column permutations can be done on the basis in order to create a form, which gives a low fill-in during the inversion.

3.2 Triangularization method

During inversion the order that we use to replace the columns of $\mathbf{I}$ should be selected carefully. As the remaining columns are updated, the number of non-zeros in $\mathbf{v_i}$ vectors can increase as a result of possible fill-ins. After an ETM is generated all remaining vectors must be updated if they have non-zero elements in the pivot row of the ETM. The other vectors remain unchanged.

Row and column counts are introduced for the basis to represent the number of non-zeros in the corresponding rows and columns. These counts can be used to identify the triangular parts of the basis. The advantage of the triangular parts is that they can be inverted directly without fill-in and without affecting the remaining part.

Figure 1 demonstrates the non-zero structure of a basis after triangularization.

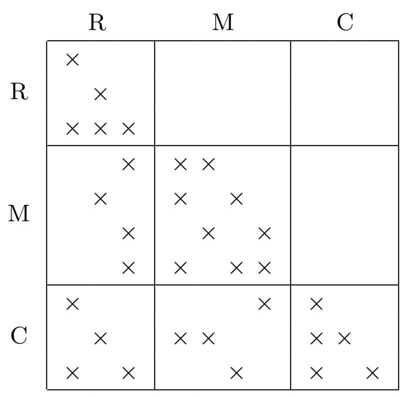

Figure 1 Triangularized form of a basis.

We emphasize that in an efficient implementation the column reordering is done only logically through a permutation vector. First, the R columns are identified and inverted. This can be done without fill-in. Next, C columns are identified. After that, the M columns must be processed (this is the most critical part of the inversion). Finally the C columns are inverted also without fill-in.

3.3 Processing the non-triangular part

The non-triangular part of the basis plays a key role in both the size and the accuracy of the inverse. We have identified three requirements to be satisfied during the processing of this part to obtain a stable inverse.

1. The number of fill-ins must be as low as possible.
2. Beside the number of fill-ins, the number of transformations must also be considered. If there is no fill-in but the non-zero elements of the basis are transformed many times with the update formula then their numerical accuracy can deteriorate. Furthermore, numerical properties must be considered. A well-scaled element can be a better pivot candidate even if it is a result of more transformations.
3. The distribution of the transformations among the non-zeros is important. The accumulation of numerical errors should be prevented or reduced. If the number of transformations is low but there are only a few elements which have been transformed many times then the numerical error accumulated on these specific values can ruin the stability of computations with the inverse.

During inversion all these properties must be taken care of.

To identify some structure in the non-triangular part (which is also sparse), we can use block-triangularization. The Tarjan algorithm [9, 4] is appropriate to identify the block structure. The result is a lower block-triangular form where non-zero elements are permuted into the diagonal. The block-triangular form can reveal hidden triangularity of the basis. This satisfies the first requirement of a stable inverse.

Each block containing more than one column cannot be inverted without numerical transformations. It is important to notice that if pivot elements are chosen within the blocks then numerical transformations must be done only on the columns of the corresponding block. So, numerical errors can not overflow from one block to another. This prevents the adding up of numerical errors throughout the whole non-triangular part thus satisfying the third requirement of a stable inverse.

After running the Tarjan algorithm threshold pivoting can be used to identify the pivot positions within the blocks. This criterion selects a subset from the eligible pivot elements. Threshold pivoting must select a pivot element within a non-triangular block satisfying

$$|v_j^i| \geq u \max\{|v_j^k|\}, \tag{5}$$

where u is an adjustable parameter and v_j is the investigated vector. The value 0.01 for u is appropriate in most cases. From the elements satisfying equation (5) one with the lowest row count is chosen for pivoting, thus the number of necessary updates is kept low.

This criterion hopefully gives an eligible numerical value for the pivot element to maintain accuracy. In special cases it can happen that the only possible choice left in a column is an element of small magnitude. Choosing a too small element as a pivot can result in an η vector with large elements because the elements of the vector $\mathbf{v}$ are divided by the pivot. If the η vector contains large elements then the small numerical errors arising from the finite precision can scale up to the magnitude of "normal" values. In the worst case the only possibility for the algorithm can be to choose a numerical garbage as a pivot element. Both can ruin the inversion procedure and lead to wrong basis changes. As a result of wrong choices after the next reinversion the simplex algorithm usually falls back to phase-1 or the objective value gets much worse.

The threshold pivoting criterion works well if the candidate set contains good elements. Therefore, the pivoting algorithm should take care of this. We have proposed and implemented a heuristic extension for processing the non-triangular part. We consider the number of non-zeros in the columns of a non-triangular block and order the columns with increasing column counts. After that the threshold pivoting procedure is called for the column with the lowest column count. Thus columns having a few non-zeros are pivoted first, preventing them to add up numerical errors on their non-zeros. Columns with higher column counts have numerical advantages in case of threshold pivoting because there is a wider range of non-zeros to be chosen for pivot elements. With this extension the balance between the number of updates and the numerical stability satisfies the second requirement of a stable inverse.

A comprehensive study on the efficiency of the reordering is shown in Section 4.

3.4 Improving numerical stability during computations

Additive operations must be performed during FTRAN, BTRAN and column updating. These operations are the main sources of numerical problems. It can happen that the difference of differently transformed, but algebraically identical quantities is a small number but not zero. In such a case the numerical garbage must be eliminated. We can introduce a relative tolerance (ε_r) and say that if the result of an additive operation is smaller than the absolute value of the larger one multiplied by the tolerance then the result can be considered zero (6). Using this technique improves the numerical properties of the algorithm.

$$a + b = \begin{cases} 0, & \text{if } |a+b| < \varepsilon_r \max\{|a|, |b|\}, \\ a+b, & \text{otherwise.} \end{cases} \tag{6}$$

During additive operations serious cancellation errors can occur. The resulting inaccurate values can scale up significantly in subsequent operations. To reduce the probability of this to happen (e.g., during computing dot products) we collect the positive and negative terms separately and add them up at the end. Similarly, we need to be cautious in case of logical (comparison) operations on two numbers. If equality is tested then absolute tolerance (ε_a) is also used. If the absolute value of the numbers difference is under the tolerance they are considered to be equal.

Both techniques have been implemented and used in our simplex solver. Now we use 10^{-10} for the relative tolerance and 10^{-14} for the absolute tolerance. The pivoting method is extended with a tolerance too, all pivot elements must be over 10^{-6} in absolute value.

4 Computational results

We have performed a computational study of our PFI implementation. For the tests we used the *netlib* collection. In Table 1 we present a subset of these problems which we use to present the findings. The chosen subset contains representatives of the sizeable and numerically more difficult problems. Problems DFL001, QAP12 and QAP15 are omitted because currently our implementation uses the primal algorithm only and these problems can be handled more effectively with the dual.

Table 1 Test problem set from the *netlib* collection.

Problem name	Rows	Columns	Non-zeros	Density
25FV47	822	1571	11127	0.86%
80BAU3B	2263	9799	29063	0.13%
BNL2	2325	3489	16124	0.19%
D2Q06C	2172	5167	35674	0.31%
DEGEN3	1504	1818	26230	0.95%
FIT2D	26	10500	138018	50.55%
FIT2P	3001	13525	60784	0.14%
GREENBEA	2393	5405	31499	0.24%
GROW22	441	946	8318	1.99%
MAROS-R7	3137	9408	151120	0.51%
PILOT	1442	3652	43220	0.82%
PILOT87	2031	4883	73804	0.74%
QAP08	913	1632	8304	0.55%
STOCFOR3	16676	15695	74004	0.02%
TRUSS	1001	8806	36642	0.41%
WOOD1P	245	2594	70216	11.04%

All the following results have been obtained on a personal computer with Intel(R) Core(TM)2 Duo E8400@3.0Ghz, 2GB RAM running 32bit Windows 7. The simplex implementation of our research group is named after the university, it is called Pannon Optimizer (PanOpt). It is important to note that the following results have been reached by using the following settings:

- Presolve techniques are not used.
- Scaling techniques are not used.
- Advanced starting basis finder techniques are not used. Every solution is obtained from the lower logical starting basis.
- Dantzig full pricing is used. This heavily effects the number of iterations needed to obtain the optimal solution.
- All these properties affect the solution time, thus our solution times published are worse than commercial solvers but they can serve as a basis for comparison. The computed optimal solutions have been validated using the COIN-OR CLP software [1].

4.1 Study of the efficiency of column reordering

In this section the numbers of non-zero transformations are analyzed. Two measurements are used for comparison. Both use the lower block-triangular form generated by the Tarjan algorithm. The measurements are done using a reinversion frequency of 60. During the tests the averages of transformations are computed. Solution times are analyzed in the next section.

The results are shown in Table 2. The first set of measurements is generated using threshold pivoting within the blocks of the block-triangular form, considering the column order given by the Tarjan algorithm. The second set of measurements is generated using threshold pivoting with the reordering of columns based on the column counts. The table also shows the reduction achieved in the number of transformations.

Table 2 Inversion statistics on average transformation numbers per inversion.

Problem name	Average number of transformations per inversion		Reduction
	Without reordering	Reordered	
25FV47	199757	130262	34.79 %
80BAU3B	386	291	24.40 %
BNL2	14782	8372	43.36 %
D2Q06C	1371770	739189	46.11 %
DEGEN3	237036	163321	31.10 %
FIT2D	4545	4033	11.26 %
FIT2P	215598	201894	6.36 %
GREENBEA	37640	15372	59.16 %
GROW22	196210	44865	77.13 %
MAROS-R7	211474	235210	-11.22 %
PILOT	11508000	10360700	9.97 %
PILOT87	47395100	36729000	22.50 %
QAP08	3859540	2387350	38.14 %
STOCFOR3	1457	1349	7.37 %
TRUSS	295365	270876	8.29 %
WOOD1P	30702	28133	8.37 %

The table clearly shows that reordering reduces the number of transformations required to compute the product form of the inverse in most cases. It can happen that the sequence of bases is different in these two cases because the numerical properties of the basis can produce numerically different reduced costs, thus the pricing strategy may choose different variables to enter the basis. This happened during the solution of MAROS-R7 in the reordered case. Bases with larger non-triangular parts have been obtained throughout the solution process.

The computation of the FTRAN and BTRAN operations become faster, too, because the resulting inverse usually has fewer non-zeros than it has in the general case. Reordering can be done very quickly because the column counts are directly available. It is important to note that the column reordering has other advantageous features. Most importantly, it provides a numerically more stable inverse, so reinversion frequency can be increased.

4.2 Investigating reinversion frequencies

In the literature the main problem with the PFI is that reiversion must be performed too frequently. Reinversion frequency of 30-60 is advised to be used. In this section we demonstrate the stability of our inversion process by increasing the reinversion frequency. This gives a substantial reduction in solution time if a problem cannot be triangularized well. Measurements have been made using both variants of the algorithm.

In Table 3 the test set with reinversion frequencies of 60, 120 and 300 and 1200 is presented without using the column ordering technique. In Table 4 the same measurements are shown with the reordering variant. The “Improvement” column shows comparison of the best solution time relative to the column of 60, which is said to be the maximal advised value. It is important to note that the column of 60 in Table 4 is already better (in most cases) than the same column in Table 3. This observation makes it clear that total improvement achieved by reordering combined with an increased reinversion frequency is more than the “Improvement”. This result is shown in the “Total improvement” column.

The results clearly show that significantly more iterations can be done without reinverting than advised. Our method seems to be numerically stable. We can solve nearly all netlib problems with the current (“work in progress”) version of PanOpt. Even PILOT and PILOT87, which are known to be numerically hard problems, can be solved with doing more than a hundred iterations in a row using the reordering variant. Remember, scaling is not used on the matrix, which proves that the numerical difficulties of the problems are handled properly by the inversion process and the FTRAN, BTRAN implementations.

Problem STOCFOR3 has a very good structure. Its non-triangular part and its average transformation count during inversion are relatively small in contrast to the problem size, thus the inversion process is fast. Unfortunately, as the sequence of ETMs representing the inverse gets longer, the BTRAN and FTRAN operations slow down and become less accurate. For such problems frequent reinversion is beneficial.

On the other hand, QAP08 and PILOT87 are extremely non-triangular, their transformation average is very high. Therefore, creating the inverse of the bases of these problems takes significant amount of time. Such problems benefit from more frequent reinversions as it can be seen in the “Improvement” columns.

It is also interesting to compare Tables 3 and 4. It shows that the reordering method generally decreases solution times. For PILOT using a reinversion frequency of 1200 solution cannot be obtained without column reordering because serious numerical problems occurred. On the other hand, with the reordered variant solution can be obtained for PILOT87 (1951.971 seconds) with reinverting only after 3000 iterations. Such results with the PFI have not been published yet.

5 Conclusions

In our work we have revisited the product form of the inverse for the simplex method. We have introduced a technique, which is appropriate to process the non-triangular part of the basis. The novelty of our approach is based on the block-triangular form. Our approach reorders the columns within each block based on the number of non-zeros in the columns. The threshold pivoting procedure is applied after reordering. In this way the resulting inverse is stable enough to be updated for hundreds or thousands of iterations and also solution times are reduced. We have implemented our method and presented a computational study to prove its efficiency.

We also have some further improvement ideas for the PFI. These ideas aim the further reduction of the number of transformations necessary to carry out inversions. Hopefully, stability will be further increased while the transformation count and solution time reduced. Any success will be reported in the future.

Table 3 Solution times with different reinversion frequencies (without reordering).

Problem name	Solution time (sec) using reinversion frequency				Best case	Improvement
	60	120	300	1200		
25FV47	15.581	12.667	18.735	48.670	12.667	18.70 %
80BAU3B	5.617	6.294	5.529	10.317	5.529	1.57 %
BNL2	9.617	12.365	14.685	63.884	9.617	0.00 %
D2Q06C	240.290	257.150	275.087	634.711	240.290	0.00 %
DEGEN3	26.811	27.299	33.402	67.905	26.811	0.00 %
FIT2D	24.774	24.619	25.271	28.135	24.619	0.63 %
FIT2P	114.350	132.001	191.432	680.007	114.350	0.00 %
GREENBEA	42.721	36.335	48.179	143.892	36.335	14.95 %
GROW22	1.067	0.947	1.024	2.178	0.947	11.25 %
MAROS-R7	28.116	23.214	24.114	37.175	23.214	17.43 %
PILOT	541.016	353.776	579.610	Error	353.776	34.61 %
PILOT87	3225.252	1988.138	1225.894	1308.544	1225.894	61.99 %
QAP08	68.090	40.026	44.149	84.112	40.026	41.22 %
STOCFOR3	58.828	63.529	87.504	217.524	58.828	0.00 %
TRUSS	35.959	23.461	31.171	60.112	23.461	34.76 %
WOOD1P	0.915	0.963	0.927	1.071	0.915	0.00 %

Table 4 Solution times with different reinversion frequencies (with reordering).

Problem name	Solution time (sec) using reinversion frequency				Best case	Improvement	Total improvement
	60	120	300	1200			
25FV47	11.604	10.957	13.553	46.733	10.957	5.58 %	29.68 %
80BAU3B	5.588	5.172	5.564	8.057	5.172	7.44 %	7.92 %
BNL2	10.576	12.386	13.770	51.131	10.576	0.00 %	-9.97 %
D2Q06C	178.310	152.192	217.553	557.962	152.192	14.65 %	36.66 %
DEGEN3	23.746	20.659	25.053	74.886	20.659	13.00 %	22.95 %
FIT2D	24.477	24.788	25.106	28.156	24.477	0.00 %	1.20 %
FIT2P	126.502	117.152	188.609	736.908	117.152	7.39 %	-2.45 %
GREENBEA	26.952	30.919	46.467	116.071	26.952	0.00 %	36.91 %
GROW22	0.858	0.928	0.948	2.150	0.858	0.00 %	19.59 %
MAROS-R7	28.019	23.887	24.230	37.412	23.887	14.75 %	15.04 %
PILOT	517.893	307.454	277.950	402.567	277.950	46.33 %	48.62 %
PILOT87	2884.267	1517.950	1089.021	1205.336	1089.021	62.24 %	66.23 %
QAP08	60.738	34.163	28.481	68.982	28.481	53.11 %	58.17 %
STOCFOR3	58.952	63.318	87.061	218.299	58.952	0.00 %	-0.21 %
TRUSS	25.518	24.662	28.309	59.582	24.662	3.35 %	31.42 %
WOOD1P	0.815	0.882	0.873	1.332	0.815	0.00 %	10.93 %

Acknowledgments We would like to thank our colleagues for their work in the implementation of other modules of Pannon Optimizer. This publication/research has been supported by the TÁMOP-4.2.2/B-10/1-2010-0025 project.

References

1 COIN-OR Linear Optimization Solver. `http://www.coin-or.org/projects/Clp.xml`.

2 G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T.C. Koopmans, editor, *Activity analysis of production and allocation*, pages 339–347. Wiley, New York, 1951.

3 George B. Dantzig and Wm. Orchard-Hays. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, 8(46):64–67, 1954.

4 I. S. Duff and J. K. Reid. An implementation of Tarjan's algorithm for the block triangularization of a matrix. *ACM Transactions on Mathematical Software (TOMS)*, 4(2):137–147, 1978.

5 A. M. Erisman, R. G. Grimes, J. G. Lewis, and Jr. W. G. Poole. A structurally stable modification of Hellerman-Rarick's P4 algorithm for reordering unsymmetric sparse matrices. *SIAM Journal on Numerical Analysis*, 22(2):369–385, 1985.

6 E. Hellerman and D. Rarick. The partitioned preassigned pivot procedure (P4). In *Sparse matrices and their applications*, pages 67–76. Plenum Press, 1972.

7 Harry M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.

8 I. Maros. *Computational Techniques of the Simplex Method.* Kluwer Academic Publishers, Norwell, Massachusetts, 2003.

9 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.