MATLAB User Guide for Plug-and-Play ADMM

Xiran Wang and Stanley Chan

Statistical Signal and Image Processing Lab, Purdue University

https://engineering.purdue.edu/ChanGroup/

Version 1.0

This user guide documents the usage of the MATLAB implementation of Plug-and-Play ADMM. Acknowledgement of this package should be given to the following reference.

[1] S. H. Chan, X. Wang and O. A. Elgendy, "Plug-and-Play ADMM for image restoration: Fixed point convergence and applications," *IEEE Trans. Computational Imaging*, in Press, Nov. 2016. ArXiv: https://arxiv.org/abs/1605.01710

1 Introduction

Plug-and-Play ADMM solves the following problem:

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\text{argmin}} \quad f(\boldsymbol{x}) + \lambda g(\boldsymbol{x}), \qquad \text{(Problem (P))}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown clean image, $f(\cdot)$ is the forward model of the image formation process, $g(\cdot)$ is a regularization function, and $\lambda > 0$ is a regularization parameter. Of interest to this package is the forward model $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2$, where $\boldsymbol{y} \in \mathbb{R}^m$ is the observed image, and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the linear transformation relating $\boldsymbol{x}$ and $\boldsymbol{y}$.

Plug-and-Play ADMM solves Problem (P) by iteratively solving the a sequence of subproblems:

$$\boldsymbol{x}^{(k+1)} = \underset{\boldsymbol{x}}{\text{argmin}} \; f(\boldsymbol{x}) + \frac{\rho_k}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}^{(k)}\|^2, \qquad \widetilde{\boldsymbol{x}}^{(k)} \stackrel{\text{def}}{=} \boldsymbol{v}^{(k)} - \boldsymbol{u}^{(k)} \tag{1}$$

$$\boldsymbol{v}^{(k+1)} = \mathcal{D}_{\sigma_k}(\widetilde{\boldsymbol{v}}^{(k)}), \qquad \widetilde{\boldsymbol{v}}^{(k)} \stackrel{\text{def}}{=} \boldsymbol{x}^{(k+1)} + \boldsymbol{u}^{(k)} \tag{2}$$

$$\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + (\boldsymbol{x}^{(k+1)} - \boldsymbol{v}^{(k+1)}) \tag{3}$$

$$\rho_{k+1} = \gamma_k \rho_k, \tag{4}$$

In this set of equations, $\{\rho_k \,|\, k = 1, 2, \ldots, k_{\max}\}$ is a sequence of internal parameters, updated by constants

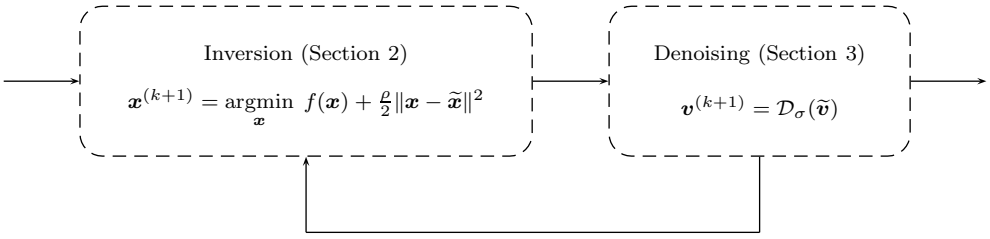


Figure 1: Block diagram of Plug-and-Play ADMM: The inversion module solves the minimization with $\widetilde{\boldsymbol{x}} = \boldsymbol{v}^{(k)} - \boldsymbol{u}^{(k)}$. The denoising module applies a denoiser to $\widetilde{\boldsymbol{v}} = \boldsymbol{x}^{(k+1)} + \boldsymbol{u}^{(k)}$.

$\{\gamma_k \,|\, k = 1, 2, \ldots, k_{\max}\}$. The function $\mathcal{D}_{\sigma_k}(\cdot)$ is called a *denoiser*, which can be any image denoising algorithm. The parameter $\sigma_k \stackrel{\text{def}}{=} \sqrt{\lambda/\rho_k}$ is the "noise level" that the denoiser takes. The denoiser $\mathcal{D}_{\sigma_k}(\cdot)$ takes the role of the regularization function $g(\cdot)$ in an implicit way. That is, given a denoiser $\mathcal{D}$ and under appropriate conditions, there exists a regularization function $g(\cdot)$ such that $\mathcal{D}(\widetilde{\boldsymbol{v}}) = \underset{\boldsymbol{v}}{\operatorname{argmin}}\ \lambda g(\boldsymbol{v}) + (\rho/2)\|\boldsymbol{v} - \widetilde{\boldsymbol{v}}\|^2$. For detailed discussions, please see [1–3]. Figure 1 shows the block diagram of the algorithm.

2 Inversion Module

2.1 Image Deblurring

Image deblurring concerns about the problem

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\ \frac{1}{2}\|\boldsymbol{H}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda g(\boldsymbol{x}), \tag{5}$$

where $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ is a circulant matrix denoting the blur. The matrix $\boldsymbol{H}$ is never formed explicitly in Plug-and-Play ADMM. It is implemented using the convolution $\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}$ with the equivalent blur kernel $\boldsymbol{h}$:

```
h = fspecial('gaussian',[9 9], 1);
y = imfilter(x,h,'circular');
```

In this MATLAB command, we assume that the blur has a circular boundary. If the boundary is not circular, one can pad the image using symmetric or replicate pads, then trim the padded boundaries after running Plug-and-Play ADMM.

The inversion step for image deblurring takes the form

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\ \frac{1}{2}\|\boldsymbol{H}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\rho}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2,$$

which has a closed form solution

$$\widehat{\boldsymbol{x}} = (\boldsymbol{H}^T\boldsymbol{H} + \rho\boldsymbol{I})^{-1}(\boldsymbol{H}^T\boldsymbol{y} + \rho\widetilde{\boldsymbol{x}}).$$

The MATLAB implementation of the closed-form expression uses Fourier transform to diagonalize the blur matrix $\boldsymbol{H}$.

```
dim    = size(y);
Hty    = imfilter(y,h,'circular'); % if h is not symmetric, do rot90(h,2)
eigHtH = abs(fftn(h, dim)).^2;
...
rhs    = fftn(Hty+rho*xtilde,dim);
x      = real(ifftn(rhs./(eigHtH+rho),dim));
```

Example. Run the demonstration file `demo_deblur.m`.

Input

Output (`method='RF'`)
31.48 dB, 0.73 sec

Output (`method='BM3D'`)
32.49 dB, 12.69 sec

2.2 Image Inpainting

Image inpainting handles the following problem

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{S}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda g(\boldsymbol{x}), \tag{6}$$

where $\boldsymbol{S} \in \mathbb{R}^{n\times n}$ is a diagonal masking matrix: $S_{ii} = 1$ if the ith sampled is selected, and $S_{ii} = 0$ if the ith sample is not selected. The non-zero entries of $\boldsymbol{S}$ can be arbitrary. Implementation of $\boldsymbol{S}$ is done using a mask. The following is an example for a random mask of 20% non-zeros.

```
mask   = rand(size(z))>=0.8;
y      = x.*mask;
```

The inversion step of image inpainting has the form

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{S}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\rho}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2,$$

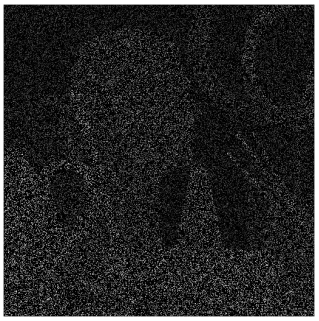
which has a closed form solution

$$\widehat{\boldsymbol{x}} = (\boldsymbol{S}^T\boldsymbol{S} + \rho\boldsymbol{I})^{-1}(\boldsymbol{S}^T\boldsymbol{y} + \rho\widetilde{\boldsymbol{x}}).$$

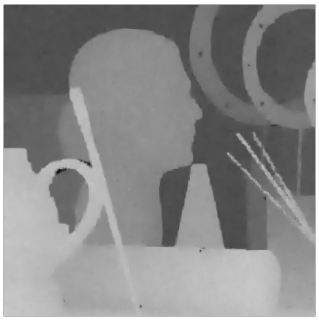
Since $\boldsymbol{S}^T\boldsymbol{S}$ is a diagonal matrix with binary entries, the closed form solution can be implemented using an element-wise division.

```
Sty    = y.*mask;
...
rhs    = Sty+rho*xtilde;
x      = rhs./(mask+rho);
```

Example. Run the demonstration file `demo_inpaint.m`.



Input



Output (`method='RF'`)
33.96 dB, 2.51 sec

Output (`method='BM3D'`)
37.38 dB, 49.91 sec

2.3 Image Super-resolution

Image super-resolution addresses the problem

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{S}\boldsymbol{H}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda g(\boldsymbol{x}), \tag{7}$$

where $\boldsymbol{S} \in \mathbb{R}^{m\times n}$, $m < n$, is a binary matrix denoting the K-fold downsampling, and $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ is a convolution matrix reprsenting the anti-aliasing filter. Both $\boldsymbol{S}$ and $\boldsymbol{H}$ are not formed explicitly in Plug-and-Play ADMM. $\boldsymbol{H}$ is implemented using convolution $\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}$ with the equivalent blur kernel $\boldsymbol{h}$ as in image deblurring, and $\boldsymbol{S}$ is implemented using the built-in MATLAB function downsample with a scale of K. The following MATLAB routine outlines the key steps in setting up the problem.

```
h = fspecial('gaussian',[9 9],1);
y = imfilter(z,h,'circular');
K = 2;
y = downsample(downsample(x,K)',K)';
```

The inversion step of super-resolution has the form

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{SHx} - \boldsymbol{y}\|^2 + \frac{\rho}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2,$$

which has the closed form solution (see the reference paper)

$$\boldsymbol{x} = \rho^{-1}\boldsymbol{b} - \rho^{-1}\boldsymbol{G}^T\left(\mathcal{F}^{-1}\left\{\frac{\mathcal{F}(\boldsymbol{Gb})}{|\mathcal{F}(\widetilde{h}_0)|^2 + \rho}\right\}\right), \tag{8}$$

where $\boldsymbol{G} = \boldsymbol{SH}$, $\boldsymbol{b} = \boldsymbol{G}^T\boldsymbol{y} + \rho\widetilde{\boldsymbol{x}}$, $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the forward and inverse Fourier transform operators, and $\widetilde{h}_0$ is the 0th polyphase component of the filter $\boldsymbol{HH}^T$. The implementation in MATLAB is as follows:

```
[G,Gt]   = defGGt(h,K);
GGt      = constructGGt(h,K,rows,cols);
Gty      = Gt(y);
...
rhs      = Gty + rho*xtilde;
x        = (rhs - Gt(ifft2(fft2(G(rhs))./(GGt + rho))))/rho;
```

where function `defGGt` defines the $\boldsymbol{SH}$ and $\boldsymbol{H}^T\boldsymbol{S}^T$ operators, and function `constructGGt` computes $|\mathcal{F}(\widetilde{h}_0)|^2$.

Example. Run the demonstration file `demo_superresolution.m`.

Input

Output (`method='RF'`)
26.51 dB, 2.16 sec

Output (`method='BM3D'`)
27.34 dB, 46.91 sec

2.4 General Linear Inverse Problem

For problems that assume a general $\boldsymbol{A}$ matrix, Plug-and-Play ADMM can be run under the general mode. A general linear inverse problem has the following form:

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda g(\boldsymbol{x}), \tag{9}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is an arbitrary matrix. In Plug-and-Play ADMM, we are mostly interested in imaging applications where m and n are very large. Therefore, $\boldsymbol{A}$ is implemented via a function handle `afun.m`.

Function Handle. The routine below is an example provided in the package. In this example, $\boldsymbol{A}$ is a blur operator with a point spread function specified by `h`. The function handle takes two options: `transp` or `no_transp`. When `no_transp` is switched on, `afun` performs the matrix multiplication $\boldsymbol{A}\boldsymbol{x}$. When `transp` is switched on, `afun` performs the matrix multiplication $\boldsymbol{A}^T\boldsymbol{x}$. The input and output to `afun` are vectors. Therefore, `reshape` function is required.

```
function y = afun(x,transp_flag,h,dim)
rows = dim(1);
cols = dim(2);
if strcmp(transp_flag,'transp')          % y = A'*x
    x = reshape(x,[rows,cols]);
    y = imfilter(x,rot90(h,2),'circular');
    y = y(:);
elseif strcmp(transp_flag,'notransp')    % y = A*x
    x = reshape(x,[rows,cols]);
    y = imfilter(x,h,'circular');
    y = y(:);
end
```

The inverse step has the form

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\rho}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2,$$

which is equivalent to

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \frac{1}{2}\left\| \begin{bmatrix} \boldsymbol{A} \\ \sqrt{\rho}\boldsymbol{I} \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} \boldsymbol{y} \\ \sqrt{\rho}\widetilde{\boldsymbol{x}} \end{bmatrix} \right\|^2.$$

Since $\boldsymbol{A}$ is implemented using function handle, we solve the above quadratic equation using LSQR (available in MATLAB). LSQR solves the quadratic equation by letting

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{A} \\ \sqrt{\rho}\boldsymbol{I} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{y} \\ \sqrt{\rho}\widetilde{\boldsymbol{x}} \end{bmatrix}.$$

Then, the algorithm calls the following routines

```
G     = @(z,trans_flag) gfun(z,trans_flag,A,rho,dim);
rhs   = [y(:); sqrt(rho)*xtilde(:)];
[x,~] = lsqr(G,rhs,1e-3);
```

The function handler `gfun` for the case of deblurring takes the original function handler `afun` and augment the function to construct the matrix $\boldsymbol{G}$.

```
function y = gfun(x,transp_flag,A,rho,dim)
rows = dim(1);
cols = dim(2);
N    = rows*cols;
if strcmp(transp_flag,'transp')         % y = A'*x
    x1   = x(1:N);
    x2   = x(N+1:2*N);
    Atx  = A(x1,'transp');
    y    = Atx + sqrt(rho)*x2;
elseif strcmp(transp_flag,'notransp')   % y = A*x
    Ax   = A(x,'notransp');
    y    = [Ax; sqrt(rho)*x];
end
```

Example. Run `demo_general.m`. The following code demonstrates the example of image deblurring using the general mode of Plug-and-Play ADMM.

```
dim = size(z);
h   = fspecial('gaussian',[9 9],1);
A   = @(z,trans_flag) afun(z,trans_flag,h,dim);
y   = A(z(:),'transp') + noise_level*randn(prod(dim),1);
out = PlugPlayADMM_general(y,A,lambda,method,opts);
```

3 Denoising Module

The denoising module of Plug-and-Play ADMM takes the form

$$\widehat{\boldsymbol{v}} = \mathcal{D}_\sigma(\widetilde{\boldsymbol{v}}), \tag{10}$$

where $\widetilde{\boldsymbol{v}}$ is the "noisy" input, and σ is the "noise level". In MATLAB, the denoising step is implemented as the function `denoise`:

```
sigma  = sqrt(lambda/rho);
v      = denoise(vtilde,sigma);
```

Inside `denoise`, the user can choose one of the following image denoising algorithms.

```
switch method
    case 'BM3D'
        denoise=@wrapper_BM3D;
    case 'TV'
        denoise=@wrapper_TV;
    case 'NLM'
        denoise=@wrapper_NLM;
    case 'RF'
        denoise=@wrapper_RF;
    otherwise
        error('unknown denoiser \n');
end
```

Remark: Plug-and-Play ADMM can support any types of image denoisers. For denoisers that are not shown in the list above, users can customize their own wrapper files to support the denoiser they use.

3.1 BM3D

The BM3D package can be downloaded at `http://www.cs.tut.fi/~foi/GCF-BM3D/`. The corresponding paper [4] is available at `http://www.cs.tut.fi/~foi/GCF-BM3D/BM3D_TIP_2007.pdf`

When using the BM3D denoiser, we call the wrapper function

```
denoise=@wrapper_BM3D;
```

where the wrapper function is defined as:

```
function out = wrapper_BM3D(in,sigma)
 [~,out] = BM3D(1, in, sigma*255);
end
```

Here, `in` is the input image, and $\sigma \in [0, 1]$ is the noise level.

3.2 Total Variation (TV)

We use `deconvtv` for total variation (TV) image denoising. The `deconvtv` package can be downloaded at `https://www.mathworks.com/matlabcentral/fileexchange/43600`. The corresponding paper [5] is available at `https://engineering.purdue.edu/ChanGroup/publication/J2011_Chan_Khoshabeh_Gibson.pdf`

When using TV denoiser, we call the wrapper function

```
denoise=@wrapper_TV;
```

where the wrapper function is defined as:

```
function out = wrapper_TV(in,sigma)
tmp  = deconvtv(in,1,1/sigma^2);
out  = tmp.f;
end
```

3.3 Non-local Means (NLM)

We use the official non-local means (NLM) package provided by IPOL. The package can be downloaded at `http://www.ipol.im/pub/art/2011/bcm_nlm/`. The corresponding paper [6] is available at the same website.

When using the NLM, we call the wrapper function

```
denoise=@wrapper_NLM;
```

where the wrapper function is defined as:

```
function out = wrapper_NLM(in,sigma)
Options.filterstrength=sigma;
out = NLMF(in,Options);
end
```

3.4 Recursive Filter (RF)

Recursive filter (RF) was originally proposed by Gastal and Oliveira in 2011, named under the framework of domain transform [7]. The original domain transform package is available at `http://inf.ufrgs.br/~eslgastal/DomainTransform/`, which includes the recursive filter.

For image denoising applications, several modifications of RF are required. These modifications include: (i) Improve robustness using patches. (ii) Recalculate weights by subtracting the noise level. (iii) Removal of cumulative sums as they are not needed. The modified RF code is provided in the Plug-and-Play ADMM package.

When using the RF denoiser, we call the wrapper function

```
denoise=@wrapper_RF;
```

where the wrapper function is defined as:

```
function out = wrapper_RF(in,sigma)
out = RF(in, 3, sigma, sigma, 3);
end
```

4 Installation

The MATLAB package includes all necessary files of the Plug-and-Play ADMM. After unzipping the package, the following files will be present:

```
./data/
./denoisers/
./utilities/
afun.m
demo_***.m
PlugPlayADMM_***.m
wrapper_***.m
```

Data

The testing data includes 9 standard testing images. In addition, there are 4 disparity images from Middlebury Stereo database `http://vision.middlebury.edu/stereo/`.

Utilities

The utilities folder contains necessary functions to support the main routines.

Denoisers

The denoiser folder contains four sub-folders:

`./BM3D/`

`./NLM/`

`./RF/`

`./TV/`

For BM3D, TV and NLM, the denoisers can be downloaded from

- BM3D: `http://www.cs.tut.fi/~foi/GCF-BM3D/`
- TV: `https://www.mathworks.com/matlabcentral/fileexchange/43600`
- NLM: `http://www.ipol.im/pub/art/2011/bcm_nlm/`

Once the denoisers are downloaded and installed, move the files into the subfolders of Plug-and-Play ADMM. An alternative option is to change path in the demo files, e.g., change `addpath(genpath('./BM3D/'))` to `addpath(genpath('C:/your_folder/BM3D/'))`.
The modified RF is included in the Plug-and-Play ADMM package. There is no need to install RF.

5 Copyright

This package is Copyright © 2016 Statistical Signal and Image Processing Lab, Purdue University.

Permission to use, copy, modify, and distribute this software and its documentation for educational, re- search and non-profit purposes, without fee, and without a written agreement is hereby granted, provided that the above copyright notice, this paragraph and the following three paragraphs appear in all copies.

The software program and documentation are supplied "as is", without any accompanying services from Purdue University. Purdue University does not warrant that the operation of the program will be uninterrupted or error-free. The end-user understands that the program was developed for research purposes and is advised not to rely exclusively on the program for any reason.

References

[1] S. H. Chan, X. Wang, and O. A. Elgendy, "Plug-and-Play ADMM for image restoration: Fixed point convergence and applications," *IEEE Trans. Computational Imaging*, 2016, In Press. Available at https://arxiv.org/abs/1605.01710.

[2] S. Venkatakrishnan, C. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *Proc. IEEE Global Conference on Signal and Information Processing*, 2013, pp. 945–948.

[3] S. Sreehari, S. V. Venkatakrishnan, B. Wohlberg, G. T. Buzzard, L. F. Drummy, J. P. Simmons, and C. A. Bouman, "Plug-and-play priors for bright field electron tomography and sparse interpolation," *IEEE Trans. Computational Imaging*, vol. 2, no. 4, pp. 408–423, Dec. 2016.

[4] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering," *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[5] S. H. Chan, R. Khoshabeh, K. B. Gibson, P. E. Gill, and T. Q. Nguyen, "An augmented Lagrangian method for total variation video restoration," *IEEE Trans. Image Process.*, vol. 20, no. 11, pp. 3097–3111, May 2011.

[6] A. Buades, B. Coll, and J. M. Morel, "A non-local algorithm for image denoising," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Jul. 2005, pp. 60–65.

[7] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph. (SIGGRAPH)*, vol. 30, no. 4, pp. 69, Aug. 2011.