BLOM: The Berkeley Library for Optimization Modeling

Anthony Kelman, Jason Kong , Sergey Vichik, Kyle Chiang, and Francesco Borrelli

Abstract— **We present the Berkeley Library for Optimization Modeling (BLOM), an open-source tool for optimization-based modeling and control formulation implemented in Simulink. The underlying structure for BLOM is a novel way of representing linear and nonlinear mathematical functions that allows for easy computation of closed form gradients, Jacobians and Hessians. This formulation provides an efficient problem representation for optimization-based modeling and is scalable to large optimization problems. With BLOM, an optimization-based controller for a dynamic system can be developed and exported from the same model that is used in forward simulation.**

BLOM is capable of solving several types of optimization problems, including static optimization problems and optimization problem with dynamics. Its intended use is for nonlinear model predictive control. We present results where BLOM is able to handle problems with tens of thousands of variables.

I. Introduction

In order to design an optimization-based controller, a researcher or engineer can choose a tool from two main groups: simulation oriented tools or optimization oriented tools. Model-based optimization is typically difficult to do with existing tools. Requirements not commonly met by modeling tools designed for conventional forward simulation include: representation of constraints and a cost function, distinguishing unknown input signals that can be freely chosen from signals with known input values, and efficient calculation of derivatives for gradient, Jacobian, and Hessian information used by optimization algorithms. There are several modeling languages that use block diagram or objected oriented approaches to describe systems. Examples include: Simulink, Modelica [1], and ASCEND [2].

Many languages and tools exist for optimization modeling [3; 4; 5], but a common weakness of existing methods is that they can be cumbersome to use for forward simulation, and model validation and verification is more difficult than in a simulation environment. The language characteristics of optimization-oriented tools can be unfamiliar and difficult to use for engineers who are familiar with system modeling but are not optimization experts, which limits collaborative model development opportunities on large scale systems. A large model that is developed using a conventional simulation and technical computing environment like Matlab or Simulink can be difficult to translate into an optimization formulation in languages like AMPL [6], GAMS [7], or AIMMS [8].

Recently the Optimica [9; 10] extension for the Modelica language was introduced. This extension facilitates the conversion of a dynamic model into an optimization problem.

In this paper, we propose a tool called the Berkeley Library for Optimization Modeling (BLOM) which bridges the gap between simulation oriented tools and optimization oriented tools. BLOM is based on a new formulation for representing linear and nonlinear mathematical functions that aims to address some of the limitations of simulation-oriented tools. This formulation allows for direct computation of closed form gradients, Jacobians and Hessians. The initial model formulation interface is based on Simulink, and BLOM provides a set of Matlab functions which transform a Simulink model into an optimization problem using our representation format. This problem representation is then used in a compiled interface to an optimization solver such as IPOPT [11]. BLOM can be expanded in the future to use the same internal problem representation with other model formulation interfaces and optimization solvers.

The primary intended use of BLOM is for nonlinear model predictive control (MPC) problems, but static optimization problems with no system dynamics or time horizon can also be represented, formulated, and solved. BLOM currently requires all functions to be in $\mathbb{C}^2$ to allow for the computation of the Hessian. Another limitation of BLOM is that it does not allow for the use of all Simulink blocks, but many commonly-used blocks are supported.

In Section II, we describe the proposed mathematical formulation used in BLOM. We then describe the Simulink interface implementation in Section III. The way that BLOM exports models for optimization problems is further elaborated in Section IV. In V we present a few problems that BLOM is capable of handling. We conclude in Section VI.

II. Internal mathematical representation

We consider optimization problems of the following form:

$$\min_{x\in\mathbb{R}^n} f(x) \tag{1a}$$

$$\text{s.t.} \quad g(x) = 0 \tag{1b}$$

$$x_l \leq x \leq x_u \tag{1c}$$

$$x_l \in \{\mathbb{R}\cup -\infty\}^n, x_u \in \{\mathbb{R}\cup \infty\}^n \tag{1d}$$

where x_l and x_u are the upper and lower bound vectors for the optimization variables x. Elements of x that are

The authors are with Mechanical Engineering Department at the University of California, Berkeley, CA 94720-1720 USA

email: {kelman, jasonjkong, sergv, kylechiang, fborrelli}@berkeley.edu

This material is based upon work supported by the National Science Foundation under Grant No. 1239552. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

This work was partially supported by the Department of Defense Environmental Security Technology Certification Program (ESTCP).

unbounded above or below have the corresponding elements of x_u or x_l equal to $+\infty$ or $-\infty$, respectively.

Consider the function, $\mathbf{y} = f(\mathbf{u})$ where $y \in \mathbb{R}^m, u \in \mathbb{R}^n$ and $f \in \mathbb{C}^2$, with the scalar input elements denoted by u_j and the scalar output elements denoted by y_j. We propose that the input-output relationship of this function can be represented in the following way:

$$0 = \sum_{k=1}^{r} K_{ik} \left(\prod_{j=1}^{n} v(u_j, P_{kj}) \right) \left(\prod_{j=n+1}^{n+m} v(y_{j-n}, P_{kj}) \right), \quad \forall\ i \in \{1, \ldots, m\}. \tag{2}$$

The parameterized function v is defined as

$$v(x,p) = \begin{cases} x^p & \text{if } p \text{ is not an exception code} \\ \exp(x) & \text{if } p \text{ is the code for } \exp \\ \log(x) & \text{if } p \text{ is the code for } \log \\ \text{etc.} & \end{cases} \tag{3}$$

We define a list of exception codes to represent transcendental functions. This list can be extended to include any differentiable single-operand function.

The matrices $K \in \mathbb{R}^{m \times r}$ and $P \in \mathbb{R}^{r \times (n+m)}$ contain, respectively, the coefficient and exponent data of a multivariable polynomial-like function. The number of monomial terms in this function is given by r. Typically the matrix P will be sparse, and K may be sparse as well.

This representation is versatile enough to represent rational functions as well. If the desired input-output relationship is $y = f(u)/g(u)$ where $f(u)$ and $g(u)$ are polynomial-like functions and $g(u) \neq 0$, this can be captured by encoding the constraint $0 = f(u) - y \cdot g(u)$ in the P and K matrices.

In addition, the formulation facilitates the calculation of closed-form gradients, Jacobians and Hessians for solvers that make use of derivative information, such as IPOPT.

In the optimization formulation, there is no need to distinguish between input and output variables (u and y in (2)). Therefore, (2) for a single row i of K can be restated as

$$f_i(\mathbf{x}) = \sum_{k=1}^{r} K_{ik} \left(\prod_{j=1}^{n+m} v(x_j, P_{kj}) \right), \tag{4}$$

where $f_i(\mathbf{x})$ is the constraint function of vector $\mathbf{x}$. The derivative of a $f_i(\mathbf{x})$ with respect to a variable x_d is

$$\frac{\partial f_i(\mathbf{x})}{\partial x_d} = \sum_{k=1}^{r} K_{ik} \frac{\partial v(x_d, P_{kd})}{\partial x_d} \left(\prod_{j \in \mathcal{J}} v(x_j, P_{kj}) \right), \tag{5}$$

where $\mathcal{J}$ is the set of variable indexes excluding d, $\mathcal{J} = \{1, \ldots, d-1, d+1, \ldots, n+m\}$. The second derivative can be computed in a similar way.

Only the non zero elements of the Jacobian and Hessian matrices are exported to a solver, because sparsity structure is immediately available from the K and P matrices. This sparse structure results in efficient performance for very large problems.

III. Simulink interface implementation

BLOM consists of three main parts. First, there is the Simulink front end, where a dynamic model is represented using built-in Simulink blocks and our BLOM library blocks. Second, a set of Matlab functions is used to convert a Simulink model into the internal mathematical representation described in Section II. Lastly, this problem representation is used by an interface to an optimization solver such as IPOPT. In the following Sections, we describe how we implement different parts of an optimization problem in Simulink.

A. Mathematical representation applied to Simulink blocks

Most commonly used Simulink blocks can be represented using the P and K matrices described in Section II. For example, an addition block with scalar inputs x_1, x_2, x_3 and output y can be represented internally in BLOM as

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

$$K = \begin{pmatrix} 1 & 1 & 1 & -1 \end{pmatrix} \tag{7}$$

where the columns of P represent the inputs x_1, x_2, x_3 and output y, respectively. Many mathematical blocks in Simulink can be represented by appropriate P and K matrices. Many commonly used mathematical blocks in Simulink are currently supported in BLOM. The P and K matrix formulation allows for the expansion of BLOM's functionality of Simulink blocks for many other Simulink blocks that have yet to be supported.

We formulate an optimization problem by introducing variables for the output signals of every mathematical block in the Simulink model, and equality constraints to enforce the input-output relationship of each block. This implicit approach increases the size of the optimization problem compared to an expression-tree approach, but the system model sparsity structure is preserved in the optimization formulation. An expression-tree approach would lead to an optimization problem with fewer variables and equality constraints, but the constraints and cost function would be denser and the calculation of gradient, Jacobian, and Hessian information would be more complicated due to nested nonlinearities.

B. Inequality constraints

Inequality constraints are marked in a Simulink model using the *Bound* block from the BLOM Library. The bound block has two user-defined parameters, *Upper Bound* and *Lower Bound*, with default values `inf` and `-inf` respectively. There are also three check boxes which specify whether the bound is enforced at the first time step, intermediate time steps, and/or the final time step of a dynamic problem.

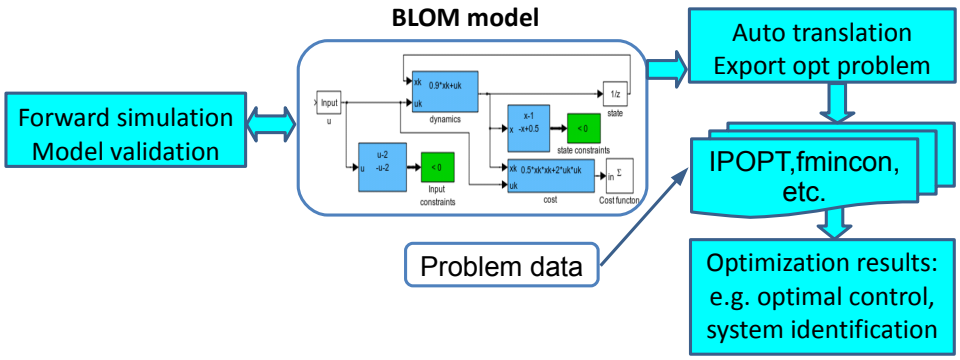

Fig. 1. Work flow with BLOM. First a model is created and validated using the BLOM library. Then, it is converted to an optimization problem and exported to one of the supported solvers.

C. Cost function

The cost function for optimization is represented in Simulink by a BLOM Library block called *Cost* with one input signal. This block does not play any role in forward simulation mode, but the input signal is treated as a cost function in an optimization problem. If the input signal is a vector, the current version sums the input elements. In the future, an option setting will be available to determine which norm to take over the input elements (1, 2, or ∞).

For models with system dynamics and time horizons longer than 1 step, the cost function is a discrete accumulator of this signal. This functionality can be further expanded to continuous integrator, the peak value of the input signal over the time horizon, or norm (1, 2, or ∞) over time of the input signal. Like the Bound block, an option allows the user to choose whether to include initial time steps, intermediate time steps, and/or final time steps in this summation or integral.

If there are multiple cost function blocks within the same BLOM model, the optimization cost function is treated as the sum of the values over all cost function blocks.

D. Control and external inputs

There are two classes of input signals to a BLOM model: unknown signals that the optimization algorithm is free to choose in order to minimize the cost function subject to constraints, and time-varying signals with a known trajectory of values over a future horizon. We will refer to the former as control inputs, and the latter as external inputs. In a MPC problem, external inputs correspond to predicted future model parameters or disturbances.

Both classes of inputs are parameterized as uniformly-sampled time series. Time variation for each input signal within one time step is important for discretization accuracy. It can be either piecewise linear and continuous (first order hold (FOH)), or piecewise constant with discontinuities at the sample times (zero order hold (ZOH)).

We assume all input signals have the same sample rate, with the exception of an optional simple implementation of move blocking on control inputs [12]. In order to reduce the number of optimization variables, the user can specify that certain control inputs should have constant values over some integer number of time steps.

In forward simulation, control input and external input blocks take input signals from non-BLOM sources in Simulink, such as Constant or From Workspace blocks. Optimization formulation only requires the dimensions of control and external input signals. The values used for external input trajectories over the optimization horizon are communicated between the Matlab workspace and the optimization solver, and do not need to be the same values as used in the Simulink block diagram.

E. Discrete-time and continuous-time states

BLOM can support both discrete-time and continuous-time models. Discrete states are implemented using the $1/z$ Unit Delay block in Simulink, and continuous states are implemented using the $1/s$ Integrator block in Simulink. If a BLOM model contains multiple Unit Delay blocks, they are all assumed to have the same sample rate. For hybrid discrete/continuous-time models, the inter-sample behavior of a discrete state block will be set to either zero order hold or first order hold as an optional field.

Continuous-time models are converted into finite dimensional optimization problems using a fixed-timestep discretization method. If there are both discrete and continuous states in the same model, the discretization step length for the continuous states must be equal to the sample time of the discrete states. The discretization method and the timestep length are specified by the user. The discretization method can be any general Runge-Kutta method, either explicit or implicit, of arbitrary order, specified by the user in the form of a Butcher tableau. Runge-Kutta methods require the introduction of additional variables for intermediate function calculations at minor time steps.

IV. Automated generation of an efficient optimization problem

As shown in Figure 1, after a model is created in Simulink and validated in BLOM, that model can be converted to an

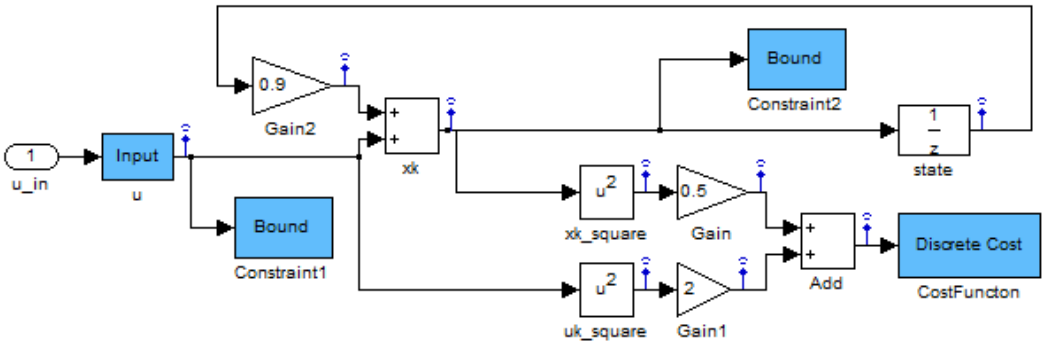

Fig. 2. Simple example of dynamic system with state and input constraints and a quadratic cost function.

optimization problem and exported to a solver. This Section details this process.

A. Model structure extraction

To extract the model, BLOM first locates the Cost and Bound blocks. From there, BLOM does a standard breadth first search that begins at these blocks and ends at the control and external input blocks. BLOM considers each scalar variable of every block found in the breadth first search as an optimization variable to consider in the formulation. Thus, for each relevant block, BLOM will generate the relevant P and K matrices relating the inputs and outputs of that block.

When the breadth first search is complete, BLOM then has a set of variables for the model in one time step. These variables are then replicated for each time step, with state variables being related between adjacent time steps to account for the propagation of model dynamics over time. This causes the P and K matrices to scale linearly across time steps and retain a sparse structure that allows for efficient computation.

B. Solver initialization and model validation

BLOM uses Simulink's forward simulation capabilities to obtain equality-feasible starting guesses for an optimization solver. Ensuring inequality feasibility may be intractable in forward simulation mode, especially for non-trivial models. Therefore, this step is not mandatory. The forward simulation is also used for model validation.

Once the forward simulation is done in Simulink for a feasible starting guess, the extracted model can be considered a standalone process and does not require the use of Simulink anymore. The remainder of the problem is done in MATLAB and IPOPT. In addition, if the model does not change, the model only needs extracted once. Once extracted, it is possible to change any external values and run the optimization problem as a standalone process.

C. Interface to optimization solvers

Once a model is created in Simulink and the optimization cost, constraints, and inputs are specified using BLOM Library blocks, the problem can be exported to an optimization solver. The interface requirements will vary depending on the solver being used, but generally a solver requires a set of user-defined functions to evaluate the cost, constraints, gradients, Jacobian, and Hessian for the optimization problem.

The BLOM problem representation allows these functions to be written in a general-purpose and straightforward way. Specialized code generation for a particular problem can be performed, but is not mandatory and in our experience code generation does not scale well to very large problems. We have implemented a compiled C++ interface to the solver IPOPT [11], which is a state of the art open source interior point solver for sparse nonlinear programming. This interface reads the P and K matrices and bound vectors from the BLOM representation for any general problem and evaluates the nonlinear functions that IPOPT requires at each iteration of its algorithm. These function evaluations contribute only a small fraction of the time required to solve an optimization problem.

V. Examples

A. Simple MPC

Consider the discrete dynamic system $x_{k+1} = 0.9x_k + u_k$ with bounded input $|u| \leq 2$ and state constraints $0.5 \leq x \leq 1$. We want to create a constrained finite time optimal control (CFTOC) problem with cost $J = \sum_{k=0}^{N-1} 0.5x_{k+1}^2 + 2u_k^2$.

Figure 2 shows a BLOM model that is used to create the CFTOC problem. The system dynamics and cost function are implemented using built-in Simulink math function blocks. Two constraint blocks (called "Bound" in Figure 2) are enforcing maximum and minimum inequality conditions. This system has a single input u that is marked by an input block to let BLOM know that it is a free optimization variable. In order to convert this model to an optimization problem, the user specifies the prediction horizon (N) and the following CFTOC problem is created

$$\begin{aligned} &\min_{u_k, x_k} \sum_{k=0}^{N-1} 0.5x_{k+1}^2 + 2u_k^2 \\ &s.t. : -2 \leq u_k \leq 2,\ 0.5 \leq x_k \leq 1, \\ &\quad x_{k+1} = 0.9x_k + u_k,\ x_0 = x(0) \\ &\quad k = 0, \ldots, N. \end{aligned} \tag{8}$$

TABLE I

BLOM WITH IPOPT PERFORMANCE ON A LARGE HVAC PROBLEM FOR VARIOUS PREDICTION HORIZON LENGTHS

Prediction horizon length	5	10	15	20	25	30
Number of variables in solver	7147	12176	18553	24846	31223	37516
Number of constraints	7329	15151	22974	30796	38619	46441
Non-zeros in Jacobian and Hessian	24057	52208	80442	108593	136827	164978
Number of solver iterations	46	230	89	82	65	68
Total solution time [sec]	2.5	8.6	26.2	48	42	97
Time spent in BLOM callbacks	11%	11%	9%	6.8%	8%	5%

B. Large scale HVAC example

BLOM is used for nonlinear MPC design of a large HVAC system [13]. The system consists of 42 thermal zones and the system dynamics are modeled with 430 state variables that represent thermal masses of elements in a building. In addition the model includes an air handling unit (AHU) model, fan model and 41 variable air volume (VAV) box models with one reheating coil each. The thermodynamic model is bilinear and additional nonlinear terms exist in the model. This system has 85 control variables that need to be determined at each time step.

Table I presents performance of the BLOM library with IPOPT solver on this problem. We present the execution time of problem preparation and solution for various problem sizes. The table shows that even for very large problems with more than 20000 variables and constraints, the library achieves good performance and IPOPT converges quickly to a KKT point of the CFTOC problem.

C. Future development

Although the BLOM library is already used in large scale industry projects, we plan to further improve and expand on its functionality for a greater scope of problems. BLOM can be developed to include support of advanced MPC techniques, such as stochastic MPC. Support of stochastic MPC will require properly handling stochastic external variables such as weather and occupancy load predictions. These stochastic variables need to be propagated properly and must be done in a way clear to the user.

There are a variety of other control problems that BLOM can be developed for, such as for use with integer problems. We believe that the core foundation of BLOM is robust and thus allows for future expansion.

VI. CONCLUSION

We proposed a new formulation for linear and nonlinear functions which allows efficient computation of closed form gradients, Jacobians, and Hessians often required by optimization solvers. The P and K sparse matrices described in Section II are portable and scalable. This formulation has proven useful when creating a Simulink library for optimization modeling.

BLOM is a good bridge between simulation modeling and optimization tools. It uniquely uses forward simulation for model validation. BLOM has yielded successful results for large scale problems with tens of thousands of variables and due to its integration with Simulink, can be easy to develop with. The ease of development helps engineers and researchers to iterate models quickly and be able to see how their system performs under different sets of conditions.

Although BLOM currently interfaces with Simulink, its underlying structure is to implement the mathematical formulation we've described given some model. Thus, it is possible to expand BLOM's functionality to include other model based designs or even have a text based structure. Interfaces to optimization solvers such as IPOPT can be reused between multiple modeling front-ends.

The most updated version of BLOM is currently available for download at `http://mpclab.net/Trac/wiki`.

REFERENCES

[1] P. Fritzson and V. Engelson, "Modelica a unified object-oriented language for system modeling and simulation," in *ECOOP98 Object-Oriented Programming*, ser. Lecture Notes in Computer Science, E. Jul, Ed. Springer Berlin / Heidelberg, 1998, vol. 1445, pp. 67–90, 10.1007/BFb0054087. [Online]. Available: http://dx.doi.org/10.1007/BFb0054087

[2] P. Piela, T. Epperly, K. Westerberg, and A. Westerberg, "Ascend: an object-oriented computer environment for modeling and analysis: The modeling language," *Computers & Chemical Engineering*, vol. 15, no. 1, pp. 53 – 72, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/009813549187006U

[3] J. Kallrath, *Modeling Languages in Mathematical Optimization*, ser. Applied Optimization. Kluwer Academic Publishers, 2004.

[4] R. d. P. Soares and A. Secchi, *European Symposium on Computer Aided Process Engineering-13, 36th European Symposium of the Working Party on Computer Aided Process Engineering*, ser. Computer Aided Chemical Engineering. Elsevier, 2003, vol. 14. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570794603802390

[5] M. Čižniar, D. Salhi, M. Fikar, and M. A. Latifi, "A matlab package for orthogonal collocations on finite elements in dynamic optimisation," in *Proceedings of the 15th International Conference Process Control '05*, J. D. J. Mikleš, M. Fikar, Ed. Štrbské Pleso, High Tatras, Slovakia: Slovak University of Technology in Bratislava, June 7-10, 2005 2005, p. 058f.pdf. [Online]. Available: http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=227

[6] R. Fourer, D. Gay, and B. Kernighan, *AMPL: a modeling language for mathematical programming*. Thomson/Brooks/Cole, 2003.

[7] R. Rosenthal, *GAMS: A User's Guide*. GAMS Development Corporation, 2008.

[8] J. Bisschop, *AIMMS Optimization Modeling*. LULU PR, 2006.

[9] J. Åkesson, *Optimica An Extension of Modelica Supporting Dynamic Optimization*. Modelica Association, 2008, pp. 57–66. [Online]. Available: https://www.modelica.org/events/modelica2008/Proceedings/sessions/session1b3.pdf

[10] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem," *Computers and Chemical Engineering*, vol. 34, no. 11, pp. 1737–1749, Nov. 2010.

[11] A. Wachter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006, 10.1007/s10107-004-0559-y. [Online]. Available: http://dx.doi.org/10.1007/s10107-004-0559-y

[12] R. Cagienard, P. Grieder, E. Kerrigan, and M. Morari, "Move blocking strategies in receding horizon control," *Journal of Process Control*, vol. 17, no. 6, pp. 563 – 570, 2007.

[13] S. C. Bengea, A. D. Kelman, F. Borrelli, R. Taylor, and S. Narayanan, "Implementation of model predictive control for an HVAC system in a mid-size commercial building," *HVAC&R Research*, 2013. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10789669.2013.834781