Chapter 10
Ubiquitous Brooks–Iyengar's Robust Distributed Real-Time Sensing Algorithm: Past, Present, and Future

This paper primarily benevolences a two-decade longstanding and most influential Brooks Iyengar's hybrid algorithm known as robust distributed computing and sensing algorithm published in IEEE computing in 1996. The algorithmic architecture establishes foundation principles across various real-time operating systems, application areas, and fault tolerant schemes. The key highlight of the algorithm is in the context of sensor applications growing interest in real-time processing and enhancing their fault tolerance characteristics of the whole system by exploiting the redundancy. The crucial contribution of the algorithm is majorly found in enhancing the features of MINIX real-time operating system, the hybrid architecture, and scalability of the algorithm is proficient enough to encounter the unreliable and distributed sensors data using the Byzantine [1] agreement and distributed decision-making process methods. This article emphasizes on inclusion and adoption of most persuasive long running Brooks Iyengar's algorithm in MINIX real-time operating system and their recent enhancement of incorporating the fault tolerant schemes. Further, the richness of algorithm has acclaimed by millions of vivid category of users around the globe in their research and computational tasks. USC/ISI. Additionally, the scalability of algorithm proved to be beneficial to other domains like cyber physical systems [3], robot merging, high-performance computing, reliability of software and hardware appliance, and artificial intelligence systems. In this paper, we attempt to showcase the use cases and real-time deployments of Brooks–Iyengar's algorithm in various aspects of distributed computational and sensor world.

The following chapter is reprinted by permission from Dr. Latesh Kumar K.J, MTS-Programmer, Computer Science and Engineering, Cloud and Cyber Security Consultant, latesh@sit.ac.in.

P. Sniatala et al., *Fundamentals of Brooks–Iyengar Distributed Sensing Algorithm*,
https://doi.org/10.1007/978-3-030-33132-0_10

10.1 Introduction: Brooks–Iyengar Algorithm

The current internet world consist numerous automated systems that must communicate with dynamic atmospheres. Because these environments are undeterminable, the systems depend on sensors for the delivery of information in order to perform the computation. The sensors are unenviable interface across computer systems and internet world (real world) for the communication of data. The smart intelligence development and deployment into these automated control systems is arduous because of limited sensor accuracy, and the noise in data readings recurrently corrupts the accuracy of data. The Brooks–Iyengar algorithm is widely known as Brooks–Iyengar hybrid algorithm [4], this algorithm acclaimed for its betterment in the accuracy of interval measurement engaged by a distributed sensor network. The key merit of this algorithm is even with the faulty sensor [5] presence the smart intelligence of sensor network swaps the measured value and precision value at each node with every peer node. Further provides accuracy in the measured range value for all the nodes of network. The resilient point of algorithm is that it is a fault tolerant and distributed, it does not malfunction even if some sensors transmit faulty data, because of this key feature it is used as sensor fusion method. Further, accuracy and precision assurance are proved in 2016 [6]. The algorithm Brooks–Iyengar integrates with Byzantine agreement with sensor fusion to control the presence of noise in sensor data. The algorithm is designed to channel the space between Byzantine fault tolerance [1] and sensor fusion. Further this algorithm is identified as the first algorithm to amalgamate these dissimilar fields. Principally, it syndicates algorithm of Dolev's [7] for an imprecise contract with fast convergence algorithm (FCA) by Mahaney and Schneider's. The core of algorithm pretends processing elements (PEs) as N and t of them are assumed to be faulty and perform malevolently. It accepts both real and unreal values with noise and uncertainty. However, the output produced by the algorithm is real value with appropriate stipulated accuracy. The algorithm is further customized to resemble Crusader's Convergence Algorithm (CCA) [8], this adoption increases the bandwidth requirement in processing of algorithm. The benefits of algorithm are wide spread across domain like high-performance computing [9], distributed control, software reliability, and real-time MINIX operating systems. The use of algorithm is not restricted to specific domains and applications we have cited. In general, all floating-point computations produce inaccuracy and this varies from machine to machine computing. This hybrid algorithm offers increased scientific consistency on a distributed system encompassing assorted components. This offers a novel method of resistance in rounding and skewing the errors generated by hardware limitations. Today's cloud based software development and customer requirements are inconsistent, the cloud based various services like software, platform, data, network, security, and recovery are different in domain but targets to produce a common service to customer. In these environments, faults tolerance and accuracy of services must be assured from end to end terms. The Brooks–Iyengar's algorithm is useful and effective in these instances by achieving robust and distributed accuracy because of the novel intelligence of algorithm. The cluster

computing involves the critical data and service modules that are important systems which demand the additional strength and accurateness. Regrettably, data, service, and security are compromised often between them, nevertheless the usage of algorithm increases without sacrificing the accuracy of data, service, and security. The fault tolerance mechanism defined in the algorithm is highly beneficial in both active and passive cluster computing in primary and disaster computing sites. With this algorithm, robust distributed computing applications can be developed and deployed seamlessly. Today's world is full of Internet of Things (IoT) and cloud based services in which sensors are vital part computing systems. The amount of data communication across current dynamic environments is leading to errors, mechanical failures, and uncertainties in sensors. To avoid this, the backup mechanism is plugged but fault tolerance and accuracy cannot be managed. Hence, the Brooks–Iyengar's algorithm has lower bound and upper bounds, and using this technique inaccuracy is dealt smartly and specifically. The algorithm is not limited to a specific domain neither restricted to set of computing and hence the wide spread smartness of algorithm is enchasing from last 20 years by various researchers, computing labs, and university training projects. In illustrating an example, a robust fault tolerant rail door state monitoring system is developed using the Brooks–Iyengar sensing algorithm to transportation applications [10], in this paper the author Buke Ao clearly listed the implementations of Brooks–Iyengar algorithm in variety of redundancy applications by various research studies [11, 12]. The key identification is Brooks–Iyengar algorithm has prominently extended seamlessly by connecting the legacy and edge cutting trends of technology variants in software applications and hardware control systems in cloud and non-cloud systems [10]. The major contribution of algorithm is identified with relevance to Linux and Android operating systems effectively. Truly, tons of various software applications and hardware control systems have encapsulated the Brooks–Iyengar algorithm to offer fault tolerant fusion data across billions of users accessing the various services through internet and other sources of digital media. Further, algorithm indirectly benefits across to 99% of world's top supercomputers and 89% of smart phones and millions of end users around the globe in various computing ways.

10.2 Real-Time MINIX Operating System

The Real-Time MINIX operating system is an enhanced version of MINIX operating system; this was originally programmed by scientist Andrew Tanenbaum for teaching operating system on $\times 86$ computer system. The research study and implementation by the author Gabriel Wainer [13, 14] changed the MINIX operating system to support RT-processing named it "RT-MINIX" by adopting Brooks–Iyengar algorithm in the areas like scheduling algorithms selection, scheduled queues, real-time metrics collection, and fault tolerant systems. Before we explain the deep impact of Brooks–Iyengar algorithm on MINIX operating system, we intend to detail about the MINIX operating system. The detailed understanding on

MINIX operating system sets a platform for understanding the Real-Time MINIX (RT-MINIX) for various applications services and control systems. The MINIX operating systems drivers, a user and system specific server runs on highest level on the miniature kernel architecture. The SYS and CLOCK are the two major tasks responsible to support the user-mode sections of the operating system at higher levels. Apart from this programming the MMU, CPU, interrupt handling, and IPC are the other privileged operations of the MINIX kernel. Just like any other operating system the functionalities like file system (FS), memory management (MM), user management (UM), and process management (PM) are offered by MINIX. The key and unique feature of MINIX over other operating system is stealthily this RS server monitors all the device drivers and various servers inside the operating systems at all time and fix it automatically when any failure is noticed.

All system calls are focused blatantly by system libraries to right server to manage the kernel communication. Let us consider a user requests a process to run an application task, usually a process is initiated by fork () library function when the process manager approves it by verifying with the memory manager on the process slots availability. If any slot available, then process manager instructs the kernel to produce a copy of the process, all this happens transparently without the notice of the user application task. Just like UNIX the MINIX kernel is responsible of managing hardware and device drivers. This involves process scheduling, interrupt management, memory, device I/O, and CPU management. The two major core components of kernel space SYS and CLOCK are explained here because in later sections we illustrate how the Brooks–Iyengar algorithm is seminal for the RT-MINIX enhancement. The SYS control is known as system task, this is vital for all kernel mode operations for the device drivers and key heartbeat channel for user segment servers. Any user request to process internally sends a signal to kernel through the library function; each request is passed to SYS. There are various categories based on the SYS management on kernel calls, to copy data between process SYS calls SYS-VIRCOPY and to configure an alarm SYS-SETALARAM etc. Few new systems call defined in the MINIX are listed below in Fig. 10.1.

The second core object is CLOCK by which kernel manages the process scheduling, timers, cron services, hardware clock, and CPU usage. The interrupt handler will initiate a timer moment when a MINIX system is power on, since then each tick is countered using this interrupt timer. In general, the cooperating servers are created by modulating the operating system; the native MINIX operating system allows third party device driver un-trusted code to run and communicate with kernel,

Kernel Call	Purpose
SYS_VIDEVIO	Read or Write a vector of I/O ports
SYS_VIRCOPY	Safe copy between address spaces
SYS_GETINFO	Get a copy of kernel information

Fig. 10.1 Privileged SYS calls to kernel

the MINIX is smart and manages the spreading of failures. A tight coupling of devices and library functions are created to intact the seamless communication in the low-level kernel operations. In this paper we are describing MINIX operating system in deep to prove how the Brooks–Iyengar algorithm is influencing the fault tolerant and robust distributed control systems in RT-MINIX.

10.3 Influence of Brooks–Iyengar Algorithm

Brooks and Iyengar's; the name is all over the globe from last two decades, this algorithm is considered to be the all-time best robust algorithm for precision, fault tolerance, and isolation of errors across software applications and hardware control systems. In this segment we narrate the Brooks–Iyengar algorithm's influence in various domains like MINIX operating system, sensor networks, software application development, real-time extensions, virtualization, physical cyber systems, and cloud computing. To begin with we explain the development and deployment of a distributed sensing algorithm that has major influence on computing systems.

10.3.1 Brooks–Iyengar's Algorithm on MINIX Operating System

The MINIX operating system powered by Tanenbaum's [15] was enhanced to Real-Time (RT) MINIX operating system and services by Wainer and it is identified as RT-MINIX [13, 14]. Further, more recent features were added to shape up an academic real-time operating system called as MINIX v2. This architecture of design was proposed to train the RTOS with few major topics:

- System architecture
- Handling interrupt
- Process management
- Scheduling of process
- Fault tolerance
- Isolation of errors

The research study by Gabriel Weiner also mentioned that many other control systems, computer application, and real-time systems are created based on the services offered by Brooks–Iyengar algorithm. The services provided by the algorithm on real-time systems, computer applications, and various systems are vaguely different from traditional systems and they are unique and different from the native operating system.

Figure 10.2 describes novel features added to MINIX operating system by Gabriel and Team in creation of RT-MINIX by using the intelligence of Brooks–Iyengar algorithm. The programming of the MINIX source code [13] was dedicated

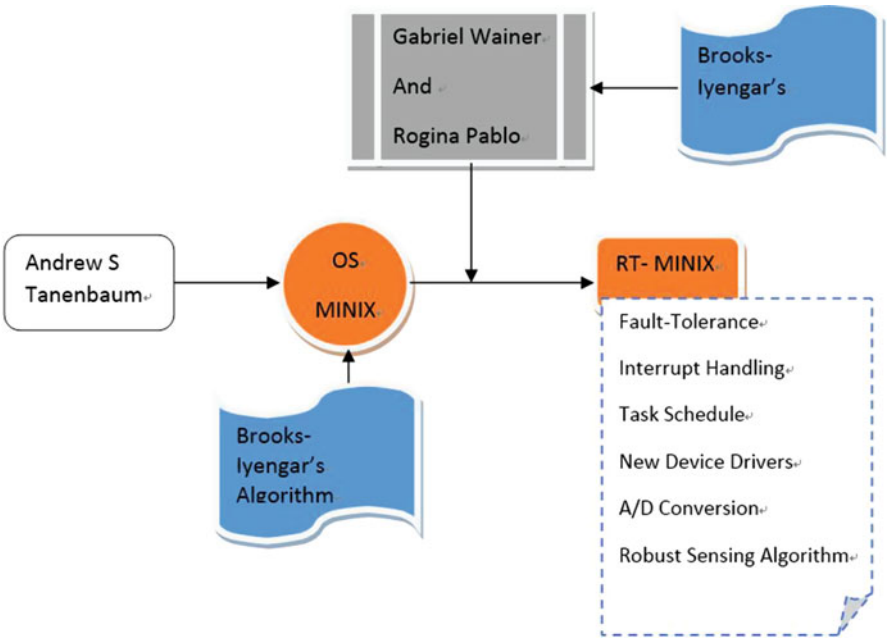

Fig. 10.2 Block structure view of Brooks–Iyengar algorithm influence

to provide the real-time controls on various services. Many real-time services were added, to begin with rate-monotonic scheduling [16], earliest deadline first processor, and fault tolerance are programmed. To make these new changes in the source code of the kernel, the code flow and data structures are slightly modified based on the new updates, specifically, sensor, timers, schedule, and criticality. Further, to adapt live-tasks with interactive CPU bound tasks a multi-queue is developed. The below listed data structure is modified in lieu of RT-MINIX evaluation (Fig. 10.3).

All these changes are tested with various feasibility of MINIX for the real-world challenges for real-time development. Numerous works were done using Brooks–Iyengar robust distributed computing algorithm from the testing of novel scheduling procedures to kernel alterations. In the meantime new version of MINIX was released and hence to sync the RT-MINIX version with MINIX version, some changes were made. The analog to digital conversion [17], in this update the target was to acquire data from analogic environment as many real-time systems are employed to handle the real process like chemical and a production line. In this requirement the Brooks–Iyengar algorithm's sensor management intelligence is effectively used for sensing the real-world data, to control the noise and to manage the faulty sensors. The interface used for game ports was used to provide the signals from the sensors, this was considered and a device driver for port is developed. The changing environment relies on poor performance of integral systems of RT-MINIX

Data Structure	Parameters	Description
struct rt_globstats {	Acrta_petsk, Act_pertsk	Period and aperiodic real time tasks, total running tasks.
int actpertsk;		Total missed deadlines
int actapetsk;	Mis_perdln,	
int misperdln;	Mis_apedln	
int misapedln;		Total real-time task scheduled instance
int totperdln;	Tot_pertsk, Tot_petsk	
int totapedln;		
int gratio;	Gratio	Guarantee ratio between deadline and instances
clock_t idletime;	Idletime	Computing Time in Second (clock Tick)
};		

Fig. 10.3 Updated MINIX data structure using the Brooks–Iyengar algorithm

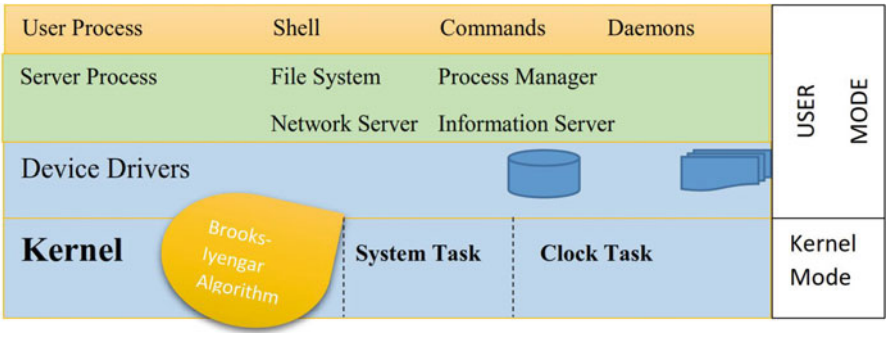

Fig. 10.4 MINIX kernel and Brooks–Iyengar algorithm

with novel techniques. The Brooks–Iyengar's algorithm adopted fast convergence algorithm (FCA) [18] to increase the convergence ratio. According the Pablo Ragina and Gabrile Weiner, the algorithm [19] is used extensively to extend RT-MINIX with possibility of several sensors from a fault tolerance perception. At the outset, the complete coding was performed based on all the four algorithms of hybrid Brooks–Iyengar. The next immediate phase was to integrate the smart capability to make use of the real-time data, to do this four potentiometers were used to sense the signals/data from analogic inputs from the joystick port. These sensor positions are arranged with actual positions for a simulation based robotic arm. An accurate and precise functionality of algorithm was noticed by providing an exclusive value from the simulated sensors in spite of faulty, at the same time users were offered open chance to modify the data by varying the potentiometers. At last, all the updated code is test for various feasibilities and real-time constraints and then the novel algorithm intelligence is united into MINIX kernel. Figure 10.4 shows the RT-MINIX Kernel and new feature additions with Brooks–Iyengar algorithm.

The software developers were given a set of functions to work with intellectual sensors, using these it was possible to generate many new services and devices like/dev/js0 and after that smart sensors were able to read data in the presence of faulty sensors. Once the operating system is enhanced with RT services, the demand ascended for various computing tools and applications. The Brooks–Iyengar's algorithm needed a test on the novel techniques applied on kernel, in order to evaluate the data structure through vivid system and library calls.

10.3.2 Case Study: Open MPI + Virtualization

The Brooks–Iyengar algorithm was further implemented on Linux using the Open-MPI [20], this is an open source project created to pass message through interface. This is a collaborative consortium of industry partners, research community, and groups of academic. Hence, the OpenMPI is powerful and smart because the knowledge, technology, and resources are shared from various communities. The libraries of MPI provide support to software developers and researchers of computer science and operating researchers.

A classical problem in distributed computing is Byzantine generals problem, introduced in the 1982 white paper of the same name. It attempted to formalize a definition for faulty (traitorous) nodes in a cluster and how for the system to mitigate it. Solutions such as majority voting or signed messages were suggested. Majority voting requires that all generals have the same information, a suggestion that is not always possible. Signed messages are a good to verify that it was the correct node in communication, even if it does not verify that the content itself is correct. Both are good suggestions, but it would be more interesting to have an algorithm that can survive a traitorous order every now and then. Enter the Brooks–Iyengar algorithm as an attempt to solve this problem. This algorithm uses sensor fusion to mathematically eliminate the faulty sensors. In short, this is achieved by taking measurements over an interval, the measured interval is then shared between all sensors on the network. The fusion step then happens, by creating a weighted average of the midpoints of all the intervals. At this point you can eliminate any sensors with high variance or use heuristics to choose the reliable nodes. It runs in $O(N \log N)$ time and can handle up to $N/3$ faulty sensors (Fig. 10.5).

To conclude the obtained results it is better to consider the dumb average because, noise generated from real and faulty sensors are from undeviating distribution. If the algorithm has not performed better then the noise would have been twisted and tremendous in one direction causing the red line curve aggressive over the green line. Overall, the algorithm is very difficult to implement as there were no framework/library and demands precision of coding and adequate infrastructure to achieve best results. The results proved that Brooks–Iyengar's algorithm is smart and scalable across various domains like cyber physical system.

OpenMPI Methods	Description
Isend and Ireceive	Non-blocking sends and receives were used to communicate from sensor to sensor. In order to process the data each sensor needs the data from every other sensor in the network. This means there are a worst case of N^2 messages being passed at any given point. Due to this large number, it is best to use non-blocking communications.
Barrier	This acts as a sync step for all sensors. Barrier merely acts as a join for processes in the context of OpenMPI. It is very useful for a simulation as this to stop one process from being a front runner.
Broadcast	Seeing as this is a timed excution program, it is necessary for each sensor to kill itself after a fixed period of time. However, it is possible for one process to keep running if it gets to the check before all the others. To get around this one thread was designated with the resposibility to check the runtime, and then broadcasted the result to all others.

Fig. 10.5 OpenMPI methods implemented using Brooks–Iyengar

10.4 Conclusion

In this article the acceleration, effectiveness, and liveliness of two decade old Brooks–Iyengar algorithm is illustrated. Since today's technology does not guarantee success and safety in all situations, the Brooks–Iyengar algorithm can significantly improve the fault tolerance of systems by providing a greater margin of safety for operations. This algorithm provides the robust implementation and seamless scalability under faulty sensor conditions for various domains. Finally, the algorithm "Stand the Test of Times" from last two decades and hope it continues the successful journey further.

Acknowledgements Authors of this book would like to thank Dr. Kumar as a prominent researcher on Storage—Cloud—Cyber-security—Protocol Engineering for the contribution to the chapter.

References

1. D. Dolev, The Byzantine generals strike again. J. Algorithms **3**(1), 14–30 (1982)
2. Penn State University, Reactive Sensor Networks, AFRL-IF-RS-TR-2003-245, Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS), Defense Advanced Research Laboratory, 2013
3. J. Park, R. Ivanov, J. Weimer, M. Pajic, S.H. Son, I. Lee, Security of cyber-physical systems in the presence of transient sensor faults. J. ACM Trans. Cyber-Phys. Syst. **1**(3), 15 (2017). https://doi.org/10.1145/3064809
4. R.R. Brooks, S.S. Iyengar, Robust distributed computing and sensing algorithm. Computer **29**(6), 53—60 (1996). https://doi.org/10.1109/2.507632. ISSN 0018-9162. Archived from the original on 2010-04-08. Retrieved 2010-03-22

5. M. Ilyas, I. Mahgoub, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems* (CRC Press, Boca Raton, 2004). pp. 254, 33-2 of 864. bit.csc.lsu.edu. ISBN 978-0-8493-1968-6. Archived from the original (PDF) on June 27, 2010. Retrieved March 22, 2010
6. B. Ao, Y. Wang, L. Yu, R.R. Brooks, S.S. Iyengar, On Precision bound of distributed fault-tolerant sensor fusion algorithms. ACM Comput. Surv. **49**(1), 5 (2016). https://doi.org/10.1145/2898984. ISSN 0360-0300
7. L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). CiteSeerX 10.1.1.64.2312. https://doi.org/10.1145/357172.357176
8. D. Dolev, et al., Reaching approximate agreement in the presence of faults. J. ACM **33**(3), 499–516. CiteSeerX 10.1.1.13.3049. https://doi.org/10.1145/5925.5931. ISSN 0004-5411. Accessed 23 March 2010
9. S. Mahaney, F. Schneider, Inexact agreement: accuracy, precision, and graceful degradation, in Proceedings of Fourth ACM Symposium Principles of Distributed Computing (1985), pp. 237–249. CiteSeerX 10.1.1.20.6337. https://doi.org/10.1145/323596.323618. ISBN 978-0897911689
10. B. Ao, Robust fault tolerant rail door state monitoring systems: applying the Brooks–Iyengar sensing algorithm to transportation applications. Int. J. Next Gener. Comput. **8**(2), 108–114 (2015)
11. V. Kumar, Computational and compressed sensing optimizations for information processing in sensor network. Int. J. Next Gener. Comput. **3**(3), 1–5 (2012)
12. B. Ao, Y. Wang, L. Yu, R.R. Brooks, S.S. Iyengar, On precision bound of distributed fault-tolerant sensor fusion algorithms. ACM Comput. Surv. **49**(1), 5:1–5:23 (2016)
13. P.J. Rogina, G. Wainer, New real-time extensions to the MINIX operating systemİ, in *Proceedings of 5th International Conference on Information System Analysis and Synthesis (IASS '99)* (1999)
14. G.A. Wainer, Implementing Real-Time services in MINIX. ACM Oper. Syst. Rev. **29**(3), 75–84 (1995)
15. S. Tanenbaum Andrew, S. Woodhull Albert, *Sistemas Operativos: Diseno e Implementacion*, 2nd edn. (Prentice Hall, Englewood Cliffs, 1999). ISBN 9701701658
16. K. Chakrabarty, S.S. Iyengar, H. Qi, E.C. Cho, Grid coverage of surveillance and target location in distributed sensor networks. IEEE Trans. Comput. **51**(12), 1448–1453 (2002)
17. B. Krishnamachari, S.S. Iyengar, Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. IEEE Trans Comput. **53**(3), 241–250 (2004)
18. S. Mahaney, F. Schneider, Inexact agreement: accuracy, precision, and graceful degradation, in *Proceedings of Fourth ACM Symposium Principles of Distributed Computing* (ACM Press, New York, 1985), pp. 237–249
19. R. Brooks, S. Iyengar, Robust distributed computing and sensing algorithm. IEEE Comput. **29**(6), 53–60 (1996)
20. Warrenedgar, An implementation of the Brooks–Iyengar algorithm using OpenMPI (2019). https://github.com/warrenedgar/brooks-iyengar