Journal of Innovations in Computer Science and Engineering
Shahid Beheshti Unversity

January 2025, Special Issue on AI 4 All- 2

Creating a Foundation for Dynamic Difficulty Adjustment within PCG of games using Imitation Learning

Navid Siamakmanesh✉, Code ORCID: 0009-0001-4010-8126
Shahid Beheshti University, Faculty of Computer Science and Engineering, Tehran, Iran, Navid.siamakmanesh@gmail.com
Mojtaba Vahidi-Asl, Code ORCID: 0000-0003-4964-992X
Shahid Beheshti University, Faculty of Computer Science and Engineering, Tehran, Iran, mo_vahidi@sbu.ac.ir
Aryan Ganji, Code ORCID:-----
Islamic Azad University North Tehran Branch, Tehran, Iran, aryanganji2003@gmail.com
Monireh Abdoos, Code ORCID: 0000-0002-3106-503X
Shahid Beheshti University, Faculty of Computer Science and Engineering, Tehran, Iran, m_abdoos@sbu.ac.ir

Abstract— This research proposes a novel approach to create a foundation for dynamic difficulty adjustment (DDA) within computer games that use procedural content generation (PCG), utilizing imitation learning to optimize gameplay. When PCG is used in creating the levels and enemies within a game, the difficulty adjustment must be ensured so that the game is not too hard or too easy for each player. However, PCG is random by nature, and thus, the developers may have a challenging task of adjusting the difficulty for each player in such games. The study aims to address these limitations by developing a foundation for DDA models based on imitation learning. The proposed model incorporates an imitation learning component, referred to as the 'Clone,' which replicates the player's behavior, alongside an enemy creator agent that leverages procedural content generation (PCG) to design enemies. By analyzing the Clone's performance against these procedurally generated enemies, the system ensures the creation of fair and engaging levels. To this end, a 2D platformer Unity game using PCG was developed, and imitation learning was utilized through Unity's ML-agents module. These models were used to mimic the players' play-style to predict the player's performance in PCG-generated levels. Three separate models were created to mimic five players. It was observed that two of these models could mimic players' performance, showing that this method can be used to implement DDA.

Keywords— Dynamic difficulty adjustment, Procedural content generation, Imitation learning, Computer Games

I. Introduction

Procedural content generation (PCG) is used in computer games to generate algorithmically and non-manually content to increase re-playability, make the game more enjoyable, and make developing the games relatively easier. Computer games often use this approach to create dynamic and diverse game worlds. PCG algorithms can be designed to generate random content or based on specific rules and constraints, which allows efficient and adjustable content generation [1]. In recent years, new methods of procedural content generation such as grammar-based, rule-based, solver-based, and search-based methods, as well as many others, have been used to generate various types of content, namely levels, maps, character models, and textures [2]. Notably, PCG in games has more usage than just entertainment and it has been used for formal and informal educational games, well [3]. Even more fascinating, PCG is often used to build a virtual environment to test and assess the ability of the algorithm to solve problems [4], and Increasing generality in machine learning algorithms [1]. Dynamic Difficulty Adjustment (DDA) is a game design technique used to automatically adjust the difficulty level of a computer game during gameplay based on the player's performance, skills, or other relevant in-game factors. Dynamic difficulty setting aims to provide a balanced and enjoyable gaming experience by making games more challenging for skilled players and easier for weaker players. This often involves adjusting parameters such as AI, enemy health, and damage power or designing levels to maintain player engagement and prevent frustration [5]. Balancing difficulty in video games is crucial to keep players engaged; games that are too easy or hard games can cause boredom or frustration, leading players to quit. Dynamic difficulty adjustment (DDA) modifies the difficulty of the game in a manner that the player always remains within the Flow channel[1][6]. enhancing experience and playtime by adjusting based on real-time player data. [7]. A simple plot of the flow channel in games can be viewed in Fig. 1:

Submit Date: 2024-12-01
Accept Date: 2025-05-05
✉ Corresponding author
[1] In games, the flow channel is a balance where challenge meets skill, keeping players engaged—neither bored nor anxious.

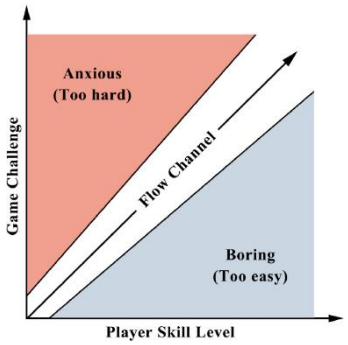

Fig. 1. Example of an appropriate difficulty level to achieve flow channel [6]

Imitation learning is a machine learning technique in which an agent learns to perform tasks by imitating the actions of an expert, using a dataset of state-action pairs. This is particularly useful when defining reward functions is challenging, and is commonly used in scenarios where an expert's behavior serves as a benchmark for task performance, scenarios where demonstrating the optimal behavior is easier than calculating the reward function, or situations in which the objective is to reduce the exploration space by learning behavioral strategies from expert trajectories. Imitation learning enables the teaching of complex tasks with limited expert knowledge [8]. such techniques can simplify task instruction by providing demonstrations, effectively eliminating the need for detailed programming or task-specific reward functions [9]. Imitation learning is divided into three general categories: behavioral simulation, inverse reinforcement learning, and adversarial imitative learning [10]. Imitation learning is also used for robotics [11], medical treatment trajectory optimization [12], and self-driving vehicles [13].

This research proposes a novel foundation for creating a DDA that uses imitation learning to clone the players' strategy and capabilities. This clone can effectively mimic the player and play Procedurally generated levels. If the difficulty of these levels is appropriate for the clone, it will be appropriate for the player. This innovative approach redefines gameplay experiences, breaking away from fixed difficulty settings. Despite the impressive progress of PCG in various game fields, not much attention has been paid to the production of non-playable characters, and most of the work done was on the production of the game maps. Also, past research rarely combines the procedural content generation element with the dynamic adjustment of task difficulty . To this end, this research focuses on creating a PCG for non-playable characters and creating clones using imitation learning to dynamically adjust the difficulty of such characters.
The outline plan contains two major elements: a "clone" model, which mimics a person's play-style and skill level, and an enemy generator agent, which uses PCG to make appropriate challenges. Most importantly, it is desired that the clone shows the same performance as the player in procedurally generated, random levels. If this task is achieved, this model can be used to give the developers of games an insight into appropriate levels and enemies for the player thereafter.
This system was implemented and tested within a 2D platformer game developed in Unity, utilizing Unity's ML-Agents module to support the imitation learning process.

II. Methodology

The proposed method focuses on the implementing imitation learning in procedurally generated game environments. This requires that each of the following parts be done:

III. To implement the desired algorithm and freedom of action in doing all the objectives, a simple game with the intended mechanics is made and used in the following steps. The intended game for simplicity will initially be just stages of fighting with a single non-playable characters that PCG randomly generates. Each fight with the aforementioned created enemy can be called a level.

IV. Since no data is available from the player the first time he plays the game, the first level of the game is generated manually. The player's performance in this level and the next levels are used to create a clone of the player. Alternatively, more levels can be used before dynamic difficulty adjustment to ensure model accuracy before implementing DDA.

V. Development of an algorithm that enables an imitative agent to predict the player's actions during the game. The Generative Adversarial Imitation Learning (GAIL) algorithm will be used to achieve this task. GAIL algorithm simple teaches an agent to imitate expert behavior by combining imitation learning with Generative adversarial network(GAN). The agent's policy acts as a generator, trying to produce actions that mimic an expert, while a discriminator distinguishes between expert and agent actions. The agent is trained to misguide the discriminator, by this method the agent essentially becomes a clone of the expert and thus learning expert-like behavior without using a reward function[14]. The GAIL algorithm can be viewd in Fig. 2:

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \quad (1)$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta), \quad \text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \quad (2)$$

6: **end for**

Fig. 2. The GAIL algorithm [14].

VI. By implementing the GAIL algorithm it is now possible to create a clone with the function CreateClone, when the demonstration of the player's behavior is present, it would be used for playing future levels. The statistics of the clone's behavior in future levels can give the developers an insight into the possible behavior of the player. After the first levels are completed the clone will play through the levels generated by the level generator to measure the difficulty of the generated level. Many different variables can be considered in determining the difficulty of the level. Examples are the win chance, the duration it takes to finish the level, and how close was the agent to being defeated. The proposed foundation framework can be seen in Fig. 3:

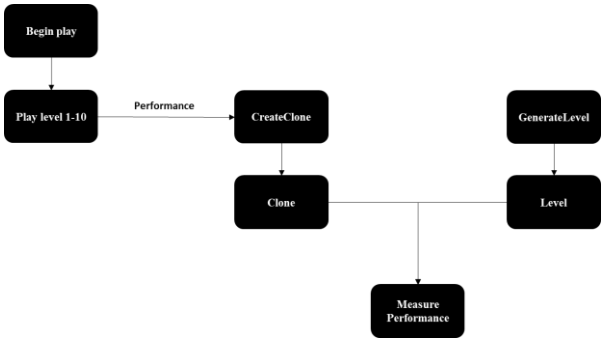

Fig. 3. The proposed foundation framework

VII. At this point the foundation for DDA is completed. With this foundation,, the developer can choose which method is used for adjusting the difficulty. They can use the clone to play the future levels and choose the level that has the closest correlation to the desired difficulty. For example, if they desire that the player finish the level with the minimum possible health they can produce several levels that would be played by the clone and choose the level that was finished closest to 0 health. Also, now that a basic clone is available it is possible to simply improve the existing clone from here on. The function ImproveClone can be created to use the already existing model and the data of the player's performance to further improve the clone to mimic the player's behavior and strategy with better accuracy in future levels. This creates a loop of the players playing the game, clones playing these levels, the level with the best difficulty settings chosen to be used by the player, the player playing this level, and in the end the clone improving by the performance of the player in that level. With all the modules completed, it is possible to release the game. A view of the proposed usage method's process can be seen in Fig. 4:

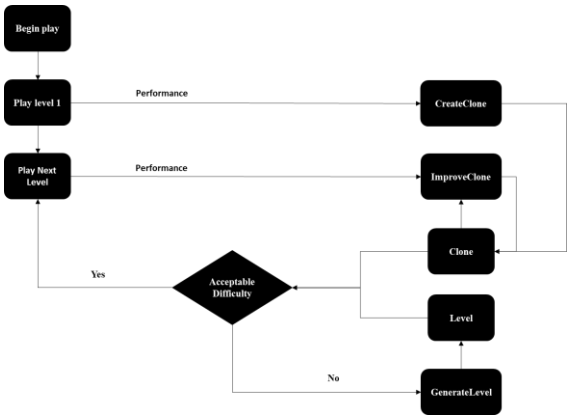

Fig. 4. The proposed usage method's process

VIII. results and discussion

The first step, as mentioned before, was creating a game with PCG capabilities that created random enemies with different sets of actions. To achieve this goal, a simple 2d game was created in Unity Environment. In this game, the enemy randomly chooses 3 out of 30 possible actions. Each action can have 4 possible difficulty levels. This creates 2.71×10^14 possible enemies. Furthermore, the starting location of the player, and moving behavior of the enemy are also randomized. Each level is a fight between the character and 1 single enemy. with this level of randomness, the task of creating the PCG is completed. A sample image from the game environment can be seen in Figure 5:

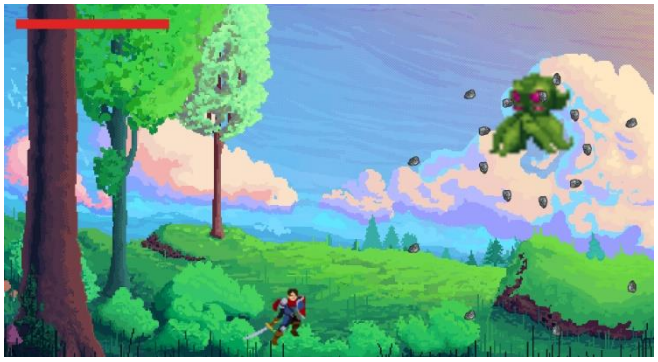

Fig. 5. A sample image from the game

The next step is to have a demonstration of players playing the game. these demonstrations would be used to train all models. To this end, five demonstrations were collected from five players. 10 scenarios with fixed enemy actions and locations were created for the players to play. This would ensure that consequent performance results are determined purely by the skill and strategy of the players and not their luck.

Players were all male between the ages of 18 and 25. Player 1 had no experience in playing games, players 2,3,4, and 5 all had a decent amount of experience in playing games in general. Player 5 had previously played this game for a short amount of time. In all scenarios, the enemy and the player had 10 health, and each time they took damage they would lose 1 health. There was no limit on time to finish the level. Their performance can be seen in Fig. 6

TABLE I. PLAYER PERFORMANCE

Players	**Players' performance**			
	Average Level-End Time	***Average Player Health***	***Average Enemy Health***	***Average Win Rate***
Player 1	27.9	0.6	4.8	0.2
Player 2	19.9	2.6	1.4	0.6
Player 3	17.2	4.5	1.5	0.6
Player 4	19.9	4.1	0.9	0.7
Player 5	17.6	7.6	0.3	0.9

Fig. 6. Average player performance across various metrics, including the average time taken by players to complete each level, the average remaining health of both the player and the enemy at the end of each level, and the average win rate (the ratio of successful completions to total attempts for each level).

A basic GAIL model was created to mimic the players. The configuration of this model is specified in Fig. 7:

TABLE II. PLAYER PERFORMANCE

Model Configurations	**Models**		
	Model 1	***Model 2***	***Model3***
Trainer	PPO		
Hyperparameters	Batch Size: 256, Buffer Size: 1024, Learning Rate: 0.0003, Beta: 0.0005, Epsilon: 0.2, Lambda: 0.99, Epochs: 3		
Network	Hidden Units in each layer: 256 Normalize: True		

Model Configurations	**Models**		
	Model 1	***Model 2***	***Model3***
	Layers: 4	Layers: 4	Layers: 8
Reward Signal (GAIL)	Gamma: 0.99, Strength: 0.8, Hidden Units in each layer: 256,		
	Layers: 4	Layers: 8	Layers: 4
Training	50 episodes		

Fig. 7. Configuration of Models 1, 2, and 3.

Each model trained on the demonstration provided by each of the 5 players, creating 15 models, i.e. clones of players. after this, each model played 50 PCG-generated levels.
Unfortunately, as it can be seen the average level end time for players 2, 3, 4, and 5 is far too close to be considered a differentiating variable for further investigation and the performance of the models in Average Level-End time would not provide further insight on the model's performance and accuracy. As such this variable will not be included in the results. However, the plot of the three variables of win rate is shown in Fig 8., Average Player Health is shown in Fig 9., and Average Enemy Health is shown in Fig 10. compared to player Performance can provide useful information in regard of investigating the model performance. The aforementioned figures are presented below:

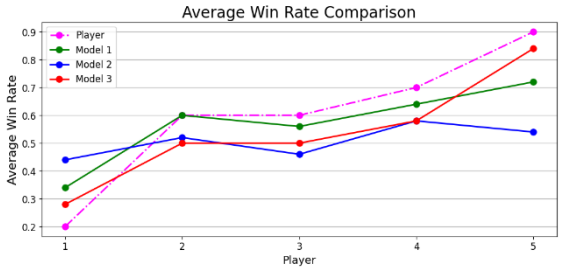

Fig. 8. Average Win-rate comparison between demonstration from players and PCG-generated levels played by Models

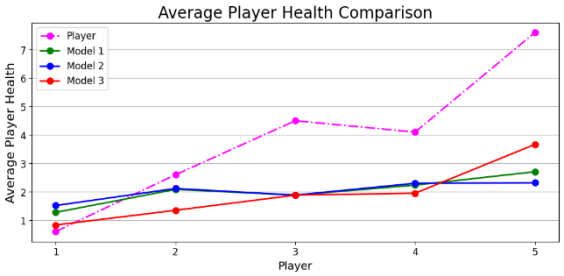

Fig. 9. Average player health comparison between demonstration from players and PCG generated levels played by Models

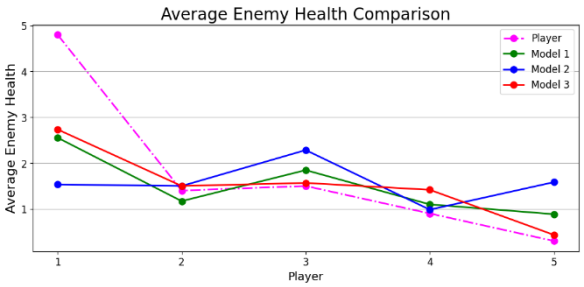

Fig. 10. Average enemy health comparison between demonstration from players and PCG-generated levels played by Models

The results show that while model 2 failed to produce and acceptable Clone to the players, the results from models 1 and 3 show a clear correlation to the results from players. both models kept their monotonic increase for win rate and player health and maintained a monotonic decrease in enemy health. Same as the real data provided by the players.
model 3 can mimic the player and maintain the same win rate as them with an average of 9.2% with a maximum of 12% error and model 1 can mimic the player and maintain the same win rate as them with an average of 8.4% with a maximum of 18% error. Model 2 unfortunately has a staggering average of 18% with a maximum of 36% error. All 3 models error in predicting player health is around 20% but their error in predicting enemy health is far less, 7.2% for models 1 and 2 and 5.7% in model 3. These results show that the created models can produce results closely correlating to real player results.
Yet it is obvious that neither model could exactly copy the players' results. There are many factors to consider regarding this problem. The most obvious reason is the randomness of the environment. Each model played 50 different levels which is not even a fraction of the total possible generated levels. The reason further tests was avoided is that these models will run during

the play of real players and training or testing can not be a straining task for the computer as it could possibly crash the game or make the loading screens far longer. Another part to consider is that there are many hyperparameters and possible architectures for the networks used in this model. Only 3 models were created with basic and common hyperparameters and architectures and 2 of them showed a good correlation to real players' performance. Signifying that with more models it is quite possible to create one that mimics the players with far less error. Another factor to consider is the players themselves. Out of the 5 players, 2 had the same win rate of 60% and another had a win rate of 70%. More tests on real players would be crucial to better understanding of the models' performance. As of the moment, this model can be a foundation for developers to use in games that use PCG to have a good grasp of the players' experience and strategy in playing their game.

IX. conclusions

In this study a new foundation for dynamic difficulty adjustment was created for games that use procedural content generation by using imitation learning. For this purpose, Generative adversarial imitation learning algorithm was used to create three different models that closely mimic players strategies and performance. A usage method for this model was proposed that can employ this foundation for dynamic difficulty adjustment. Also, to have an environment to test these models, a 2d platformer simple game was created in unity platform. In the created game elements of PCG was used to create random levels and test the model's capabilities. 5 players created demonstrations of playing fixed levels and their results were used to create these models. Each of the models played 50 PCG-generated levels and their performance was compared to real players' performance. 2 of these models proved capable of mimicking players' behavior with a close average win rate of 9.2% and 8.4%. these results show that mimicking the player's performance is possible with this method.

X. Limitations and Future Work.

This work only created the foundation for dynamic difficulty adjustment. The possible usage for this foundation was proposed but is not yet created. By replicating the proposed method's process the task of improving said model can be advanced.
Despite the randomness of the levels generated by the game, the models showed acceptable results in mimicking players strategies and performance. However, a number of possible solutions exist for minimizing the error further. In this project, only 5 players volunteered to play the game and 3 of them showed similar results. In order to have a better insight into model's performance it is necessary for more players to create demonstrations of their play.
Each model played 50 random levels but if more levels are played the accuracy of the models can be pinpointed more specifically.
There are many hyper parameters architecture specifications in the created model. These parameters can be fine-tuned in order to create better models that could replicate the players' behavior with greater accuracy.

Acknowledgment

The authors would like to express their gratitude to the players who participated in this study, contributing their valuable time by playing the game and providing demonstrations.

References

[1] S.Risi . and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nat Mach Intell*, vol. 2, no. 8, pp. 428–436, 2020, doi: 10.1038/s42256-020-0208-z.

[2] J. Liu, S. Snodgrass, A. Khalifa, S. Risi, G. N. Yannakakis, and J. Togelius, "Deep learning for procedural content generation," *Neural Comput Appl*, vol. 33, no. 1, pp. 19–37, 2021, doi: 10.1007/s00521-020-05383-8.

[3] M. İnce, "BiLSTM and dynamic fuzzy AHP-GA method for procedural game level generation," *Neural Comput Appl*, vol. 33, no. 15, pp. 9761–9773, 2021, doi: 10.1007/s00521-021-06180-7.

[4] J. P. Sousa, R. Tavares, J. P. Gomes, and V. Mendonça, "Review and analysis of research on Video Games and Artificial Intelligence: A look back and a step forward," *Procedia Comput Sci*, vol. 204, pp. 315–323, 2022, doi: 10.1016/j.procs.2022.08.038.

[5] T. Huber, S. Mertes, S. Rangelova, S. Flutura, and E. Andre, "Dynamic Difficulty Adjustment in Virtual Reality Exergames through Experience-driven Procedural Content Generation," *2021 IEEE Symposium Series on Computational Intelligence, SSCI 2021 - Proceedings*, 2021, doi: 10.1109/SSCI50451.2021.9660086.

[6] Q. Mi and T. Gao, "General Dynamic Difficulty Adjustment System for Major Game Genres," *Lecture notes on data engineering and communications technologies*, pp. 189–200, Jan. 2023, doi: https://doi.org/10.1007/978-3-031-35836-4_21.

[7] P. Moschovitis and A. Denisova, "Keep Calm and Aim for the Head: Biofeedback-Controlled Dynamic Difficulty Adjustment in a Horror Game," *IEEE Trans Games*, vol. 15, no. 3, pp. 368–377, 2023, doi: 10.1109/TG.2022.3179842.

[8] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud Resource Scheduling with Deep Reinforcement Learning and Imitation Learning," *IEEE Internet Things J*, vol. 8, no. 5, pp. 3576–3586, 2021, doi: 10.1109/JIOT.2020.3025015.

[9] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput Surv*, vol. 50, no. 2, pp. 1–35, 2017, doi: 10.1145/3054912.

[10] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation Learning: Progress, Taxonomies and Challenges," *IEEE Trans Neural Netw Learn Syst*, pp. 1–16, 2022, doi: 10.1109/TNNLS.2022.3213246.

[11] K. Kutsuzawa and M. Hayashibe, "Imitation Learning With Time-Varying Synergy for Compact Representation of Spatiotemporal Structures," *IEEE Access*, vol. 11, no. March, pp. 34150–34162, 2023, doi: 10.1109/ACCESS.2023.3264213.

[12] S. I. H. Shah, A. Coronato, M. Naeem, and G. De Pietro, "Learning and Assessing Optimal Dynamic Treatment Regimes Through Cooperative Imitation Learning," *IEEE Access*, vol. 10, no. June, pp. 78148–78158, 2022, doi: 10.1109/ACCESS.2022.3193494.

[13] W. J. Yun, M. Shin, S. Jung, S. Kwon, and J. Kim, "Parallelized and randomized adversarial imitation learning for safety-critical self-driving vehicles," *Journal of Communications and Networks*, vol. 24, no. 6, pp. 710–721, 2022, doi: 10.23919/jcn.2022.000012.

J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," *arXiv:1606.03476 [cs]*, Jun. 2016, Available: https://arxiv.org/abs/1606.03476