Evolving behaviour trees for automated discovery of novel combat strategy in real-time strategy wargames

Martin Masek[a], Chiou Peng Lam[a], Luke Kelly[a] and Martin Wong[b]

[a] School of Science, Edith Cowan University, Joondalup, Western Australia, [b]Defence Science Technology Group, Department of Defence, Australia
Email: m.masek@ecu.edu.au, martin.wong@dst.defence.gov.au

Abstract: A wargame is defined as: "A simulation, by whatever means, of a military operation involving two or more opposing forces using rules, data, and procedures designed to depict an actual or assumed real life situation" (Gortney 2016, p503). Wargaming is used for several purposes, such as teaching strategic planning, practising the tasks associated with war, and analytic purposes (Burns et al., 2015). While wargaming is predominantly a human centric analytical activity, it is an area where artificial intelligence (AI) may play a useful role because of a computer's ability to play through a wider range of possible strategies. However, creating simulated AI participants for wargame scenarios is challenging because of the complexity and uncertainty in the environments in which they exist. The recent rise of automated behaviour discovery for agents in a variety of games traditionally dominated by humans offers a possible solution. Commercial real-time strategy (RTS) games provide an abstract simulation of a world where players aim to dominate and defeat other players by acquiring, using and managing resources, often including a mix of military, political, scientific and economic factors. As there is considerable overlap between the objectives found in RTS games to those that exist in military style wargames, RTS games are ideal platforms to conduct research and development in support of our AI-enabled wargaming research objectives.

In this paper, we document our work on the use of evolutionary algorithms for automated behaviour discovery, where evolved behaviour trees are used as controllers for the blue team entities in the wargaming simulation. As behaviour trees are constructs formulated as a tree-type graph data structure, genetic programming (Koza, 1992), a technique developed specifically for the evolution of such structures was employed. In order to systematically evaluate the approach in terms of novel behaviour discovery, two test scenarios were designed to isolate particular features of land-based combat, inspired by terrain design patterns from computer games. A more complex scenario, involving multiple terrain constructs was also evaluated.

A set of experiments were run on the developed scenarios to evolve a blue team against a static red team opponent. The red entities employed a reactive AI, with the simplicity of the red AI balanced by having a much greater number of red units in the scenario. Results from the experiments indicate that evolved behaviour tree controllers in a multi-agent scenario can be useful to identify a set of behaviours that exploit the properties of the scenario and lead to victory. In particular, we observed the 'expected' behaviour in the control scenarios:

- Units learned to work together as a team, for example, armor units shielding artillery units.
- Units in a relatively weak blue side learned to exploit a chokepoint in the terrain to defeat a superior red side.
- Units learned to avoid dangerous sections of terrain, such as those protected by enemy snipers.

Keywords: *Wargames, evolution of behaviour trees, genetic programming*

1. INTRODUCTION

Behaviour trees (BT) are constructs that have been used to model opponents for strategy games (Colledanchise & Ögren, 2018). They are data structures where nodes of the tree represent conditional checks and actions of an in-game entity (typically agent-based) and are widely used in the creation of hand-coded game AI. The actions and transition logic are separated into action and control flow nodes respectively. Complex behaviours can be built by arranging control flow and action nodes into a tree structure, which is traversed depth-first to determine which actions to execute. As the control flow logic is separated from the actions, new types of actions can be added to existing behaviour trees without affecting the rest of the tree. The behaviour tree also provides flexibility in that multiple instances of action nodes can be included in the tree. Behaviour trees can be built at numerous levels of abstraction with a hierarchical structure where an action node may be an entire sub-behaviour tree.

The transparent nature of behaviour trees, whereby the decision points leading to executed actions are distinctly identifiable in the tree, make them ideal to applications in behaviour modelling for wargaming and RTS games. For example, the Norwegian army used manually crafted behaviour trees to model behaviour, such as battle drills, for computer generated forces (Evensen et al. 2018). Work has also been done to automatically construct behaviour trees from observations of RTS gameplay (Robertson & Watson, 2015).

Evolutionary Algorithms (EAs) have in the past been used to automatically construct agent controllers. For example, we have used a genetic algorithm to evolve finite state machine- based aircraft controllers in an air combat scenario (Masek et al. 2019). EAs are search and optimisation algorithms based on the concept of survival of the fittest. A major defining characteristic is that they work with a population of possible solutions rather than being focused on optimising a single solution instance. They have the properties of being able to scale to large, complex search spaces and are invariant to the fitness landscape and as such can cope with discontinuities and local optima. This gives evolutionary algorithms an advantage in warfare scenarios where a range of very different strategies can lead to success, the success of each strategy is dependent on an opponent's capability to adapt and where there is no simple linear ranking of strategies as there is an element of intransitivity in them.

Previous work on evolving behaviour trees has been done in the Mario platformer environment, where Perez et al (2011) used grammatical evolution. In an application closer to war gaming, Berthling-Hansen et al. (2018) evolved behaviour trees to model the movement of a simulated soldier in a 3D military simulation environment. Evolution was done through a multi-objective genetic algorithm (NSGA-II), but changes were made to incorporate genetic programming into the algorithm. Follower behaviour was evolved using action and condition nodes based on the moving towards and querying distance to a target.

Other work in the evolution of behaviour trees has mostly been focused on combining sets of hand-coded trees. Lim, Baumgarten and Colton (2010) used a BT agent to play the game DEFCON. They hand-coded various basic BTs, then used genetic programming to combine and evolve these trees to produce a complete agent. Hoff and Christensen (2016) attempted to create a BT agent for the game Zero-K. They used genetic programming to generate a BT for each aspect of the game (unit recruitment, economy control, construction, military control etc.) and stitch these together to make a holistic agent. Due to computational constraints, they were only able to produce BTs to control the economy.

In this paper, we present an approach using genetic programming (Koza, 1992), an EA designed for the evolution of trees, to evolve behaviour trees for wargame-like scenarios. The rest of this paper is organized as follows: In Section 2 we outline our approach to behaviour tree evolution, followed by the description of our test scenarios in Section 3. The experiments and results section of Section 4 focuses on the observed behaviour from the scenarios, followed by a conclusion in Section 5.

2. AN APPROACH FOR EVOLUTION OF BEHAVIOUR TREES FOR WARGAME SCENARIOS

A high level overview of the evolutionary approach is given in Figure 1. Inputs to the system include 1) the behaviour tree primitive node set, comprising of the set of action and condition nodes that the agent controllers will be constructed from, 2) a fitness function that is used to compute a single fitness value for each solution based on the outcome of its evaluation and 3) a scenario comprising of a map layout, including the placement of blue and red teams.

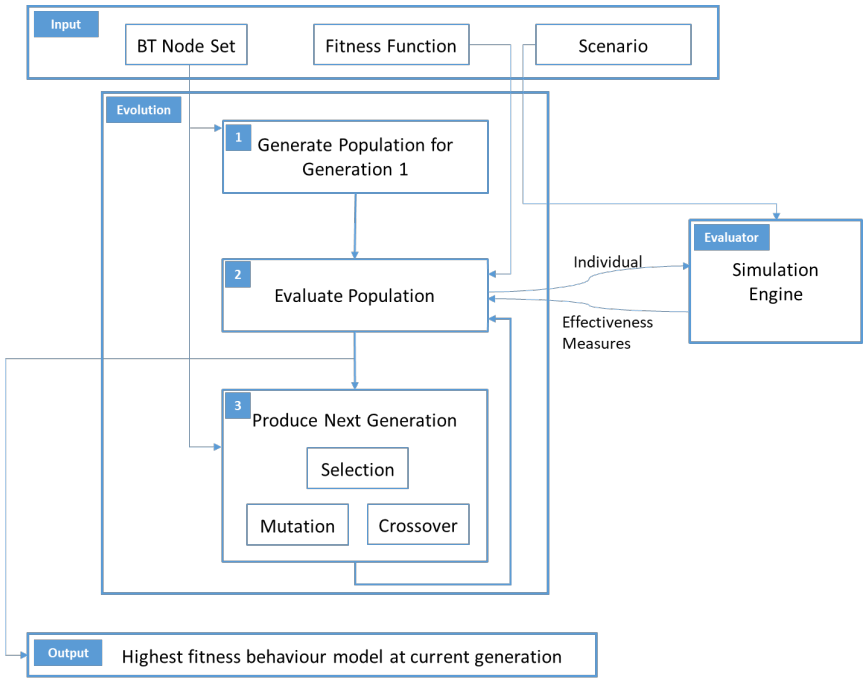

Figure 1. High level architecture of the evolutionary system using MicroRTS[1] as the simulation engine

2.1. Agent Model

A behaviour tree is constructed from a primitive set of nodes which is the collection of allowed terminals and non-terminals that our system can use to build the behaviour tree that controls the BT agent. The primitive set of behaviour tree nodes that we have used in our experiments is listed in Table 1. The terminal nodes correspond to either actions or conditions that map to the chosen evaluator (MicroRTS). The non-terminal nodes are standard behaviour tree constructs and not unique to our application.

Table 1. Primitive set used to build behaviour trees in our experiments**.**

Terminal Nodes	**Description**
AttackClosestEnemy	A command is issued for the agent to attack the closest enemy fighting unit. If there are no enemies, this node returns a failure. If there are enemies, this node returns a success.
AttackClosestBase	A command is issued for the agent to attack the closest enemy base unit. If there are no bases, this node returns a failure. If there are bases, this node returns a success.
MoveToClosestAllies	A command is issued for the agent to move towards the closest allied unit. If there are no allies, this node returns a failure. If the command to move towards the closest ally was executed, this node returns a success.
Idle	A 10 frame wait command is issued to the agent. This node always returns true.
CheckForEnemies	Returns true if enemy fighting units are present on the map, otherwise returns false.
CheckForAllies	Returns true if an ally is within a distance of 2 units, otherwise returns false.
CheckForBases	Returns true if an enemy base is present on the map, otherwise returns false.
Non-Terminal Nodes (Arity of 1-4)	**Description**
Selector	Executes child nodes in order until one succeeds.
Sequence	Executes child nodes in order until one fails or all are visited.

[1] Available from: https://github.com/santiontanon/microrts

For the red team agents, we utilise agent controllers that come as part of MicroRTS. For all experiments, we used a 'rush' style AI, where each fighting units attacks the closest enemy unit that has an accessible path to it.

2.2. Evolutionary Approach

Population initialisation
The initial population is generated using a "ramped half and half" approach, with parameter settings shown in Table 2. This produces a set of trees to a predefined maximum tree depth using two methods, grow and full. The grow method is used to initialise half of the population; it randomly constructs an individual tree from a set of non-terminal nodes, until a single non-terminal reaches the max tree depth minus one and, a terminal node is placed to meet the maximum depth. The remaining non-terminal nodes are populated with children from the terminal set. These terminal nodes must achieve a predefined minimum tree depth. The full method fills the remaining half of the initial population by generating trees where each terminal node is at the maximum tree depth.

Fitness function
A simple fitness function based on the number of unit hit points remaining at the end of the scenario. When a unit comes under fire, the damage incurred is modelled as a loss of hit points, with the unit destroyed and removed from the game when its hit points reach zero. At the end of the game, the total number of hit points remaining for the red and blue teams, redHP and blueHP respectively are calculated by summing the hit points remaining for all remaining units on a particular side. From this, the fitness of the blue team (which is the fitness assigned to the behaviour tree individual being used as the blue agent controller) is calculated using equation 1.

$$\textbf{\textit{Fitness}} = blueHP - redHP + maxHP \quad \textbf{\textit{(Equation 1)}}$$

redHP = remaining red team HP at the end of the simulation.
blueHP = remaining blue team HP at the end of the simulation.
maxHP: Highest starting team HP for the scenario.

This is similar to the fitness function used by Gajurel et al. (2018), who used the equivalent of difference in red team hit points between the start and end of the game plus the total blue team hitpoints as a measure of fitness.

Table 2. Population initialisation parameters for the ramped half and half approach.

Parameter	Value
Grow Probability	0.5
Full Probability	0.5
Min Tree Depth	2
Max Tree Depth	6

Selection, Mutation and Crossover
The selection scheme used is the Stochastic Universal Sampling (SUS) algorithm (Baker, 1987). SUS is a proportionate based selection process that favours chromosomes with a higher fitness score, but also ensures that chromosomes with a lower fitness score have a chance of making it into the following generation.

Subtree mutation, a standard mutation method in genetic programming, was the method selected as the mutation operator. The subtree mutation process is to select a random mutation point on the existing tree and replace the selected node (and its subtree) with a randomly generated subtree. To generate the subtree we used the grow method, as discussed in the population initialisation section.

The crossover scheme used is single point subtree crossover, a standard crossover method in genetic programming. Two parents are required to perform single point crossover, selected using SUS. A random point on each parent tree is selected as the place where they swap their respective subtrees, producing two new child trees.

3. SCENARIOS

We demonstrate our approach using three scenarios, plain terrain, chokepoint and urban warfare, depicted in Figure 2. The plain terrain and chokepoint scenarios were inspired by the work of Hullett & Whitehead (2010), who identified several 'design patterns' in first-person shooter levels. As the optimal behaviour in these two scenarios is known, they provide an objective means of evaluation for our approach. The third scenario, urban warfare, was chosen as a larger, more complex scenario to further explore the discovery of novel behaviour using our approach. The scenarios will now be detailed.

Plain Terrain

This terrain is based on the arena design pattern (Hullet & Whitehead, 2010), consisting of a wide open area, shown in Figure 2 (left). In this scenario, we attempt to evolve a strategy for the 'numerically inferior' blue force where the terrain is fully accessible. Each side has a base (in opposite corners of the map). Blue has one heavy unit and one ranged unit, red has five light units. In this scenario, where the blue side is outnumbered, according to Hullet & Whitehead (2010), the optimum strategy is to "move deliberately and make use of cover". Since no cover is provided in this level (apart from the blue units themselves) deliberate movement and coordination is the focus.

Chokepoint

The chokepoint design pattern (Hullet & Whitehead, 2010) is a narrow path between two areas which must be traversed to get from one area to the next. When passing through a narrow chokepoint, the power of a large force is reduced as the chokepoint limits the size of the front, meaning only a limited number of units can participate in combat. This allows a side with lower numbers to engage a numerically superior opponent on a more even basis. In this scenario, we attempt to evolve a strategy for the 'numerically inferior' blue force where the terrain features a chokepoint. The scenario, depicted in Figure 2 (middle) has red and blue sides separated by mostly impassable, apart from a narrow area, the chokepoint. Each side has a base (in opposite corners of the map), blue has one heavy unit and one ranged unit and red has five light units.

Urban Warfare

This larger (32x32) scenario, depicted in Figure 2 (right), features impassable terrain intended to symbolise urban buildings. The blue forces begin the mission at the bottom of the map. The map features a large red base in the top right-hand corner, a pool of red reinforcements at the bottom left-hand corner, with terrain designed so that these reinforcements meet the blue forces some way into the scenario. The path down the middle of the map is heavily defended by red 'snipers' – ranged units behind the cover of terrain. Due to the terrain, the red snipers are invulnerable to blue light and heavy units. The red snipers can only be targeted by blue ranged units, but the blue ranged are far outnumbered. A relatively safe path to the large red base exists on the left-hand side of the map and contains along it a small red base defended by two red heavy units. This scenario investigates a larger, more complex scenario in comparison to the previously listed scenarios, composed of several design patterns. Interesting features include the choice of a path (sniper alley versus safe flanking route) and the small number of blue ranged units which are the only units capable of eliminating the red snipers.

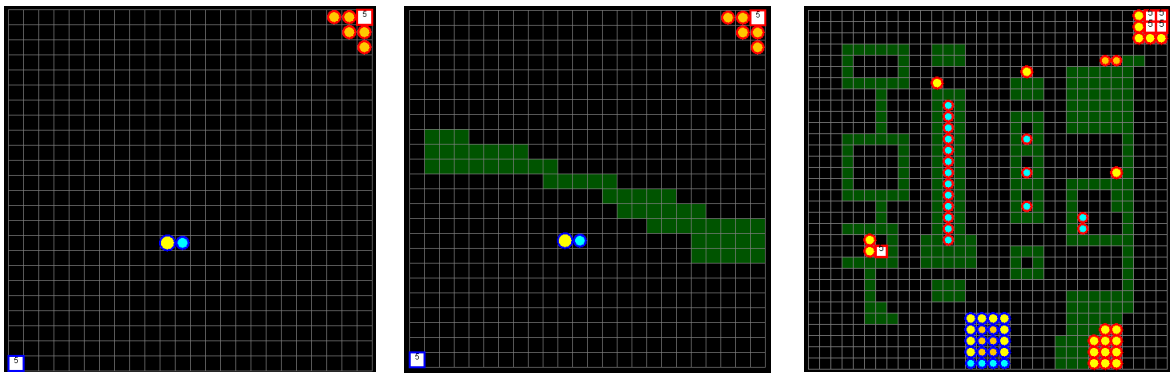

Figure 2. Scenarios implemented include Plain Terrain (left), Chokepoint (middle), and Urban (right).

4. EXPERIMENTS AND RESULTS

This section provides a summary of the behaviour observed in each of the experimental scenarios. For each scenario, multiple evolution runs were undertaken, with varying probabilities of the mutation [0.2, 0.4, 0.6,

0.8] and crossover operators [0.4, 0.6, 0.8] in each run to vary the evolution process. This resulted in twelve combinations of settings, and thus twelve runs of evolution for each scenario. In each run, evolution occurred from an initial population of 100 random solutions for 20,000 generations in the case of the plain terrain and chokepoint scenarios, and 10,000 generations for urban warfare (number of generations lowered to decrease computation requirements). The behaviour of the fittest solution in the final generation in each run was examined.

In each of the plain terrain experiments, the first action of the blue team is to avoid the red team and destroy the red base. The blue ranged unit then stays close to the wall, which stops it from being surrounded. The blue heavy gets into position between the blue ranged and the red team protecting the blue ranged unit. The blue ranged unit uses its ability to attack from a distance to eliminate the red ream, while the heavy offers protection and distracts the red units.

This experiment found the pairing of the heavy unit, with the ranged unit to be an effective team composition on plain terrain. The heavy unit can take some damage, while the ranged cannot, this makes it a capable defending unit. The heavy performed best when it didn't attack and concentrated on distracting the red team by continuously moving. The blue ranged unit was found to be ineffective by itself as it did not survive for very long if the heavy was eliminated. The ranged unit's ability to attack from a distance made it successful in eliminating the majority of the red team when it had protection.

The algorithm was successful in finding a solution to the plain terrain scenario. Of the twelve experimental runs, eight resulted in a win for the blue team and, four resulted in a stalemate situation where each team had surviving units at the end of the simulation. No runs resulted in an outright loss for the blue team.

In the chokepoint scenario, the dominant evolved tactics typically started with the blue heavy entering the chokepoint to block it so the red team could not funnel through. This action protects the blue ranged unit and allows it to eliminate the red units stuck in the chokepoint. The heavy would push out of the chokepoint and use either the left-hand wall or the green wall on the right to not allow itself to become surrounded. The heavy proceeds to move and distract the red team, while the blue ranged eliminates them from a safe distance. After the elimination of the red team, they would then destroy the red base. Other not as successful tactics that emerged saw the blue ranged attack over the impassable wall or for the blue units to wait for the red units to move through the chokepoint towards their base before attacking. This tactic was not as successful as the smaller blue team did not utilise the chokepoint and would get overrun and lose.

The chokepoint scenario experiment had similar findings to the plain terrain scenario, where the pairing of the heavy unit, with the ranged unit, was seen to be an effective team composition when the heavy defended the ranged and the ranged acted as the offensive unit from a safe distance. The ranged unit was found to be ineffective by itself as it would get overrun quickly. The algorithm was successful in finding a solution to the chokepoint scenario. Of the twelve experiment runs, all resulted in a win for the blue team.

In the urban warfare scenario, a dominant evolved tactic involves the blue units staying in a tight formation and waiting out of reach at the start of the sniper path for the red forces to attack them. The blue units engage as a team, and when the majority of the attackable red units are eliminated, some of the remaining blue team take the safe path and destroy the red bases.

The urban warfare experiment showed caution on behalf of the blue team as they avoided going into areas where the red snipers could attack them. Blue ranged units also used the impassable terrain to their advantage by firing over and attacking enemy unit while they were protected. The algorithm was successful in finding a solution to the stronghold scenario. All twelve experiment runs resulted in a win for the blue team.

5. DISCUSSION AND CONCLUSION

This work has explored the potential of evolutionary algorithms to be used in the discovery of novel behaviour in combat scenarios. The advantage of the evolutionary approach is the ability to search a large multi-modal search space and through a number of runs of the evolution find plausible solutions. The disadvantage is the computation time, the majority of which is taken up evaluating candidate solutions thorough simulated scenario runs. For example, a single run of the plain terrain or chokepoint experiments (20,000 generations) was completed in approximately seven hours, whereas the urban warfare scenario (10,000 generations) completed in approximately 200 hours. The computational requirements result in having to cap the number of generations evolved, whereas even better solutions may be found through further evolution.

The particular approach explored was the genetic programming of behaviour trees that are used as agent controllers in a multi-agent simulation. The approach was successfully demonstrated against a range of fixed scenarios, able to produce strategies that beat the built-in evaluation agent AI which was controlling a force superior in numbers and fire power. In the small-scale flat terrain and chokepoint scenarios, the known optimal behaviour was successfully produced, and in the larger scale urban warfare scenario, a winning strategy was able to be produced.

ACKNOWLEDGEMENTS
This research is supported by the Defence Science and Technology Group, Australia under the Modelling Complex Warfighting Strategic Research Investment. We would like to thank Matthew Burr who worked on the implementation of the approach.

REFERENCES

Baker, J. E. (1987, July). Reducing bias and inefficiency in the selection algorithm. In Proceedings of the second international conference on genetic algorithms (Vol. 206, pp. 14-21).

Berthling-Hansen, G., Morch, E., Løvlid, R. A., & Gundersen, O. E. (2018, June). Automating Behaviour Tree Generation for Simulating Troop Movements (Poster). In 2018 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA) (pp. 147-153). IEEE. https://doi.org/10.1109/COGSIMA.2018.8423978.

Burns, S., Della Volpe, D., Babb, R., Miller, N., & Muir, G. (2015). War Gamers Handbook: A Guide for Professional War Gamers. Naval War College Newport United States. https://doi.org/10.21236/AD1001766

Colledanchise, M., & Ögren, P. (2018). Behavior Trees in Robotics and Al: An Introduction. CRC Press.

Evensen, P., Stien, H., & Helge Bentsen, D. (2018). Modeling Battle Drills for Computer-Generated Forces using Behavior Trees. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, Florida, November 2018.

Gajurel, A., Louis, S. J., Méndez, D. J., & Liu, S. (2018, August). Neuroevolution for RTS micro. In 2018 IEEE Conference on Computational Intelligence and Games (CIG) (pp. 1-8). IEEE.

Gortney, W. E. (2016). Department of Defense Dictionary of Military and Associated Terms (No. JP-1-02). Joint Chiefs of Staff Washington United States.

Hoff, J. W., & Christensen, H. J. (2016). Evolving Behaviour Trees:-Automatic Generation of AI Opponents for Real-Time Strategy Games (Master's thesis, NTNU).

Hullett, K., & Whitehead, J. (2010, June). Design patterns in FPS levels. In proceedings of the Fifth International Conference on the Foundations of Digital Games (pp. 78-85). ACM.

Koza, J. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press.

Lim, C. U., Baumgarten, R., & Colton, S. (2010, April). Evolving Behaviour Trees for the Commercial Game DEFCON. In European Conference on the Applications of Evolutionary Computation (pp. 100-110). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-12239-2_11

Masek, M., Lam, C.P., Kelly, L, Benke, L. & Papasimeon, M. (In press 2019), A Genetic Programming Framework for Novel Behaviour Discovery in Air Combat Scenarios, Proceedings of ASOR/DORS 2018.

Perez, D., Nicolau, M., O'Neill, M., & Brabazon, A. (2011, August). Reactiveness and Navigation In Computer Games: Different Needs, Different Approaches. In Paper presented at the 2011 IEEE Conference on Computational Intelligence and Games (CIG'11), Seoul, South Korea, August 31st-September 3rd 2011. (pp. 273–280) IEEE. https://doi.org/10.1109/CIG.2011.6032017

Robertson, G., & Watson, I. (2015). Building Behavior Trees from Observations in Real-Time Strategy Games. 2015 International Symposium On Innovations in Intelligent Systems And Applications (INISTA) https://doi.org/10.1109/INISTA.2015.7276774