Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 131–140
DOI: 10.15439/2023F6015
ISSN 2300-5963 ACSIS, Vol. 35

Genetic Algorithm for Planning and Scheduling Problem – StarCraft II Build Order case study

Konrad Gmyrek, Michał Antkiewicz, Paweł B. Myszkowski
Wrocław University of Science and Technology
Faculty of Information and Communication Technology
ul.I.Łukasiewicza 5, 50-371 Wrocław, Poland
Email: {konrad.gmyrek, michal.antkiewicz, pawel.myszkowski}@pwr.edu.pl

***Abstract*—The Planning and Scheduling (PS) problem plays a vital role in several domains, such as economics, military, management, finance, and games, where finding the optimal plan and schedule to achieve specific goals is essential. In this article, we present a Genetic Algorithm for the Planning and Scheduling (GAPS) problem in the StarCraft II Build Order Optimization problem (SC2 BO) context – as it signifies that modern strategy games present a more challenging environment than classical planning problems. We evaluate the performance of GAPS and compare it with state-of-the-art methods. Experimental results provide valuable insight into the effectiveness of GA in the context of the PS Problem under various configurations, notably in the context of Lamarckianism and the Baldwin Effect. Ultimately, this research enhances the understanding of GA application for the PS problem, offering notable insights regarding GA performance and potential for future work.**

I. Introduction

A *PLANNING* problem involves creating a series of strategic steps to achieve a specific goal. It often encapsulates order and conditions under which various actions must be performed while simultaneously factoring uncertainty, resources, and goals of a given problem instance. *Scheduling* is a decision-making process that manages the allocation of resources over time. Its primary objective is to optimize specific performance metrics, typically total completion time, often called *makespan*. Therefore, planning and scheduling problems could be described as determining the optimal sequence of actions to achieve specific goals in the most time and resource-efficient manner. Despite a long history of research into these problems, the need for solutions persists due to the consistent emergence of new challenges and their increasing complexity. Planning and scheduling apply to diverse real-world applications, including manufacturing, pathfinding, logistics, management, space exploration, telecommunications, economics and games (i.e. economic or computer ones).

This article aims to solve the planning and scheduling problem in the context of the economic development aspect in the StarCraft II (SC2) strategy game, in this area known as *Build Order Planing*.

In the following sections, we will provide the state-of-the-art analysis (Sec.II), define the exact problem (Sec.III), and present the proposed approach (Sec.III) and experimental results (Sec.IV) that describe the application of Genetic Algorithm (GA) to *Build Order Planing*. The last sections of the paper consist of conclusions (Sec.V) and future works (Sec.VI).

II. Related Work

A. Build Order-Planning

As described in the [5], Build Order Planning or Build Order (BO) optimization is a class of Automated Planning problems that arise in a video game genre called real-time-strategy (RTS). It is a process of finding the sequence (order) of actions to be made by the player to achieve a specific goal in the shortest makespan. This goal depends on the exact scenario and larger strategy, but it can be associated with creating a certain number of military units or gathering the precise amount of resources. Individual RTS games differ, but the basics are mostly similar. The player performs a series of actions like collecting resources, developing a base, and training military units, all in order to gain an advantage over the opponent and finally defeat him. Mentioned actions can be divided into two groups: strategical level actions (*macro*) and tactical level actions (*micro*). In this paper, we discuss a sequence of the macro actions only - called Build Order.

In the context of the SC2 game, simple build-order solutions, such as a quick attack with a small number of military units, may be countered easily by building defensive structures whose overall cost is lower compared to the cost of the attack. Therefore, engagement benefits the defending side. On the other hand, a defensive stance is disadvantageous against economic development, as it will overtake the defending player in the long term. Finally, development focus can be easily countered by an aggressive strategy (known as 'early rush'), which closes the cycle. Based on this stone-paper-scissors-like dynamic, both players can change their focus during a game to gain an advantage against the opponent. Selecting goals and switching them in-game time is a separate subject that can be described by a high-level strategy Artificial Intelligence (AI) system as described in [1]. We can imagine it as a sort of manager that dictates the overall strategy goal, however, the problem of determining the plan for achieving this goal in the most efficient manner is the subject of the presented research.

RTS games are interesting application domains for AI researchers because of the huge state spaces and concurrent actions. The most common instance of BO application in games is StarCraft (published by Blizzard Entertainment), a

popular RTS game with over 10 million copies sold. StarCraft has received over 50 industry awards, including over 20 "Game of the Year" awards. It is considered the so-called e-sports game, as each player has the same chances at the beginning, does not contain random elements, and the result depends solely on the player's skill. As StarCraft (and its sequel - StartCraft II) is the most commonly used case, multiple other RTS games are known for the importance of early game build order. Such games as Age of Empires or Company of Heroes are worth mentioning. For the all mentioned real-time games, an important factor is the speed of action execution and fast reaction time. However, this is not an essential aspect of defining the BO problem. Turn-based games also make a good example.

BO-solving methods can be used to support players or design demanding opponents for games. It can also help with game balancing and exploit detection. For instance, when there are several civilizations (or races) to choose from, they should compete on a similar level. The existence of a specific action sequence that makes a civilization significantly stronger than the others is considered a product defect. The ability to simulate and detect it at an early stage of development is a significant improvement compared to the process of lengthy manual tests common in the game-development industry.

Authors of the [6] suggest that historically studied economic games such as Prisoner's Dilemma (where tit-for-tat derives) or Cournot Production games are far more streamlined in comparison to modern RTS games. In a broader look at this application, simulations of systems implemented in games refer to economic mechanisms known from the real world. An exemplary generalization can be interpreted as the optimization of the company's development strategy or even the investment strategy, where subsequent decisions depend on the results of previously taken steps. To support this assumption, Cobb-Douglas (CD) model presented in [14], has been applied by Webel et al. in [6] to the modern RTS game StarCraft: Brood War, where worker units gather resources, build infrastructure, and eventually lead to the construction of combat units. The winner of the game is the last one with a standing structure.

As suggested in [1], BO is an instance of a temporal planning problem. Temporal planning recognizes that actions take a certain amount of time to execute and acknowledges that multiple actions can occur concurrently under specific circumstances. For example, certain actions may have specific temporal requirements, such as waiting for a resource to become available or ensuring that one action finishes before another can start. By treating BO as a temporal planning problem, the planning process becomes more realistic and aligned with real-world scenarios than in the classical planning approach, which is streamlined, instantaneous, and does not consider actions interacting with each other.

To further emphasize treating planning in RTS games as a formal problem, Buro et al. in [1] present a study on the optimization of build order strategies in real-time strategy (RTS) games, highlighting the potential of using the Planning Domain Definition Language (PDDL) for modeling these problems. PDDL presented in [8] is a language utilized in automated planning competitions to standardize the description of planning domains and problems, enabling diverse planners to compete against each other. The authors of [1] also discuss the issue of concurrent execution and propose efficient mechanisms for ordering actions within the build order domain of RTS games. However, it is worth noting that even the most recent version of PDDL does not support object creation or deletion, which is essential in RTS games where object creation plays a vital role. While it is possible to simulate them implicitly within the language, it signifies that modern strategy games present a more challenging environment than classical planning problems.

Wei et al. in [3] were the first to address the BO problem as Planning and Scheduling problem with Producer/Consumer constraints. This attempt to mathematically model and approximate the BO problem is further supported by Blackford et al. article [5] about build order optimization in a multi-objective approach.

B. Methods

Several approaches to solving planning and scheduling problems exist in the literature, such as Ant Colony Optimization [13], Graph Search [5, 2], Stochastic Search [11], Evolutionary Algorithm [4, 2] and Machine Learning [9, 3].

Churchill et al. in [5] introduced a depth-first branch-and-bound algorithm (BnB) to address initial build orders in StarCraft. The authors implemented a tree search, with the root representing the initial game state, and conducted a depth-first exploration to identify an optimal solution. This solution aims to meet the goal within the shortest possible makespan. While it is possible to validate found build orders in the game environment, this approach is time- and resource-intensive. To address this challenge, the authors suggested developing a StarCraft economic simulator to measure the value of build orders effectively.

El-Nabarawy et al. in [11] implemented a Monte Carlo Tree Search (MCTS) based on a StarCraft mini-game called BuildMarines the goal is to find build orders that can provide players with a larger amount of Marines military units in a fixed amount of time. Both methods have significantly lower action space, and although they may be effectively applied to smaller problems, they differ from the efficiency and speed of evolutionary algorithms and machine learning techniques. The comparison of these methods has been described in the experiments section to support this claim further.

In the context of Evolutionary Algorithms (EA), Justesen et al. in [2] has successfully implemented the evolutionary-based method of Continual Online Evolutionary Planning (COEP) for build order optimization. The goal was to create a game agent that can, in-game time, find build orders that counter opponents' movements, also utilizing the BO simulator for evaluation. The fitness function is based on a heuristic that can describe how desirable founded BO is. Heuristic values short-term rewards higher than long-term rewards, which are

very important in game planning agents because long-term build orders can provide the agents with optimal strategy and powerful armies, which are easier to counter by opponents.

Blackford et al. implement a multi-objective evolutionary algorithm (MOEA) detailed in their work [4] for the Total Annihilation RTS game. The multi-objective approach is advantageous because solutions evolve to satisfy different goals, and the algorithm does not find just one solution but a set of different solutions, each potentially providing different strategic advantages.

Another significant implementation of a machine learning-based game agent was achieved in the form of AlphaStar [9] for StarCraft II. This sophisticated AI has the capability to generate valid build orders dynamically during gameplay.

Liu et al. in their work [10] used a reinforcement learning technique to create a low-cost StarCraft II agent and suggested that on a larger scale, this technique could create a better agent using fewer resources. Although machine learning-based methods show immense potential in solving BO problems, they require considerable resources and time to develop a functioning model and, once created based on a problem, cannot be easily applied to another similar problem. Conversely, Genetic Algorithms (GA) only necessitate game data and a build order evaluation method to produce valid build orders. It makes GA a more universally applicable solution for these types of problems. GA can also be deployed within build orders for game balancing, which is a complex problem.

For the above reasons, we implement the Genetic Algorithm in our research. As for reference methods, we propose those mentioned earlier, specifically the ones proposed by Justesen et al. as cited in [2], the COEP method configuration (COEPc), and the method proposed by Blackford et al. in [4], MOEA method configuration (MOEAc). In addition, we will present an application of the Genetic Algorithm in comparison to approaches based on Branch and bound (BnB) and Monte Carlo Tree Search (MCTS) methods, as cited in [5, 11].

III. Problem Definition and proposed Method

A. Planning and Scheduling Problem Definition

We generalize the SC2 BO problem using Planning and Scheduling (PS) problem with Producer/Consumer constraints based on [4] work.

The planning aspect of the problem involves determining which actions need to be performed to achieve a certain goal G. The scheduling aspect, on the other hand, describes the timing of action execution with consideration to available resources R and minimizing the overall completion time of all actions $a \in A$, called makespan. Solution $s \in SP$ where SP is a solution space could be described by a vector of actions a as follows:

$$s = [a_0, a_1, ..., a_{n-1}] \tag{1}$$

where n is the solution size. The solution s changes the initial game state S^a into desired state of S^x (see Eq.2).

$$S^a \xrightarrow{a_0} S^b \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} S^x \tag{2}$$

Every considered action a_x has a set of certain prerequisite conditions of execution. These prerequisites are described by resources $r \in R$. Wei et al. in [3] for the PS problem, distinguish two categories: Consumable, Renewable; and four types of resources: Consume, Produce, Borrow, and Require.

- **Consumable**
  - Consume - action can consume a specific amount of resources, for example, the cost of creating a wall (structure) in SC2 is: 100 gold, and 50 wood, so the action of making a wall would consume 100 gold and 50 wood.
  - Produce - action usually after completion provides a game environment with some kind of good (produces it). This good can be a soldier, building, research, or any other environmental element that changes the game state, but generally, we can define it as some kind of resource.
- **Renewable**
  - Borrow - action can borrow certain resources during execution time, for example in SC2 game, the action of creating a Zealot army unit needs to use Gateway building, Gateway can produce only one unit at a time, so it is borrowed by action to create Zealot action.
  - Require - a type of resource that multiple actions at the same time might require. For example, action mine coal borrows one miner and requires the mine to execute. However, in mine multiple miners can work so the resource is shared and now borrowed.

Note that given resources can belong to two or more types. For example, the SC2 Gateway building can be borrowed and produced. The list of resources associated with an action can be defined as a_R. For instance, in the action of creating a military unit a, a_R consists of consuming 100 gold and borrowing barracks for the production of the unit as follows: $a_R = \{100\ gold, barracks, null, military\,unit\}$

B. Constraints

To consider the build order a *feasible* solution, it is imperative that the set of constraints C is satisfied, which also limits SP.

Cumulative Producer/Consumer constraint refers to Consumable resources. It requires that when two actions are planned to occur simultaneously, the combined resources they need should not exceed the available quantity of those resources. For example, creating two Zealots (units) at the same time requires 200 crystals, 2 Gateways, and 4 supply consumption. Constraint can be expressed as:

$$\forall_{a \in s}\ a_R^t \leq T^t \tag{3}$$

where s is solution, a_R^t are resources required by the a action in time t and T is the total amount of resources available in time t.

Disjunctive constraint refers to borrowed resources, it states that for a pair of actions competing for execution at the same time. The constraint is satisfied when these actions do not overlap, and thereby the integrity of the system process is maintained. Constraint can be expressed as:

$$\forall_{(i,j)\in s^2, i\neq j}\ (i_{end} \leq j_{start}) \vee (j_{end} \leq i_{start}) \tag{4}$$

where pair $(i, j) \in s^2$ is every pair of two different actions that can be extracted from s. As a_{start} and a_{end}, we describe the start and end time of executing action a, respectively.

Exist constraint refers to Required resources. It states that certain resources must exist in order for action to execute. An example of an existing constraint in the context of SC2 could be Stalker army unit which we can not build until we build Cybernetics Core building. Constraint can be expressed as:

$$\forall_{a\in s}\forall_{r\in \text{exist}(a_R)}\ a_{start} \leq t \leq a_{end} \wedge r \in T^t \tag{5}$$

where $exist(a_R)$ is a subset of action a resource set a_R that contains only resources which must exist in order to execute action a. Resource r is not consumed.

According to the above definition, the problem could be defined as:
Given: $< G, R, C, A, S^a >$,
interpreted as follows. To find: Solution s that fulfills goal G in the shortest possible makespan, starting from state S^a, using resources R, considering only actions present in A, satisfying all constraints C.

C. SC2 BO problem as PS problem

SC2 is a strategic game that allows players to embody one of three distinct races, each differing in units, structures, and strategies. For our research purposes, we have selected the Protoss race, an advanced, alien-like race within the game. We denote the initial state S^a as a typical SC2 start state for the Protoss race, which includes the following resources: 50 crystals, 1 Nexus, 12 Probes, and a supply capacity of 15. Probes function as worker units that accumulate crystals at the Nexus base. These Probes have the capability to construct buildings, such as the Pylon, which not only increases the supply capacity but also permits the construction of more specialized buildings within its energy field. One such building is the Gateway, which facilitates the creation of basic military units, like the Zealot. In this context, we categorize actions like creating a Probe, constructing a Pylon or Gateway, or building a Zealot as macro actions, which is typical for SC2 BO. While we take into account all possible macro actions of Protoss race in our research, we will not delve into each one individually due to their sheer number.
To present SC2 BO as the PS Problem, the set of metrics $m \in M$ should be defined as follows:

- m^0 - total completion time,
- m^1 - total build cost of workers,
- m^2 - total build cost of the army,
- m^3 - total research cost,
- m^4 - total cost of defensive buildings.

The goal G within the SC2 BO context is represented as an *objective vector* $O = [m^1, m^2, m^3, m^4]$ comprising non-negative integer value. Assigning values on the vector positions defines exact scenarios. Each value within this vector can range from 0 to the maximum possible value of each metric m. However, m^0 is the total completion time of all actions in solution s extracted after BO simulation, which will be discussed later.

Based on each metric defined in vector O, we can divide the action space A into four distinct subsets, each containing actions that contribute solely to one specific objective metric value. This arrangement allows the given method to utilize every action in the game during the optimization process.

To address the scheduling aspect of the issue, we adopt a model based on the rule that each action is executed as swiftly as possible, under the condition of all prerequisites are met. For instance, if the task involves constructing two Zealots, this task will be executed concurrently if a sufficient amount of resources is available. It is the responsibility of the method to order actions in such a way that the arrangement is optimal in the context of the addressed rule. This model is adopted by a SC2 game simulator that conducts BO simulations. To acknowledge the need for the delay before executing certain actions, we propose *None* action, which will force a delay for the next action execution by a fixed amount of seconds.

The given definition allows the construction of solution $s \in SP$ (alternatively referred to as BO), where SP is the solution space, described by an ordered list of integers, each of which maps to a specific BO action in SC2 (an example presented in Figure 1).

In this paper, we engage with the complete action space of the Protoss race, which comprises 59 actions plus one additional *None* action. Each race in the game encapsulates actions space A of around 60 actions. In the case of the Protoss race, which we choose as the base for our research, the initial game state permits the execution of four possible actions. Notably, certain actions can expand the list of potential actions, making the explicit determination of the solution space a considerable challenge. Nevertheless, it's possible to estimate the entire set of potential genotypes by utilizing the length of the build order. This concept can be formally encapsulated in the following equation:

$$x = a^n \tag{6}$$

Here, n represents the size of the build order, a stands for the total size of the action space, and x signifies the complete number of genotypes – a similar representation is presented in [12]. For a comprehensive problem definition, particularly within the context of a BO problem, we need to define the size of the build order solution n. We will address size n in Sec.IV-B.

Finally, we define the evaluation function E, which value is to be minimized by the optimization method, as a function of end state S^x and objective vector O provided by simulator

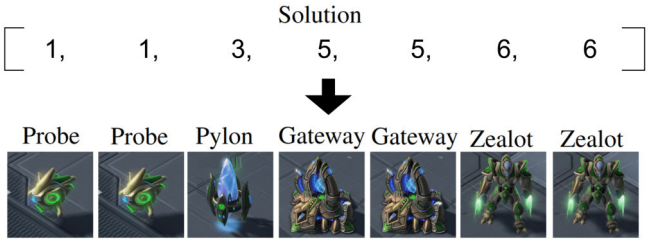

Fig. 1. The example solution – genotype (i.e. solution) presented as list of integer is mapped to list of SC2 actions

from all actions a in a given solution s. If the solution does not fulfill the objective described by the objective vector, we add a penalty for every metric m objective that is not fulfilled. E can be presented as the formula:

$$E(S^x, O) = S^{x^{m^0}} + \sum_{m \in M, m \neq m^0} max(0, O^m - S^{x^m}) \quad (7)$$

where $m \in M$ is metric, O is objective vector and S^{x^m} is end game state metric m acquired by simulation. The goal value of metric m, which is part of objective vector O can be formulated as O^m. As $S^{x^{m^0}}$ we understand the game end state time metric m^0 value. The $max(a, b)$ function represents the maximum of a and b. For example, for a given objective vector:
$O = \{m^1 = 0,\ m^2 = 200,\ m^3 = 0,\ m^4 = 0\}$
and end state S^x:
$\{m^0 = 120,\ m^1 = 800,\ m^2 = 100,\ m^3 = 100,\ m^4 = 0\}$
is evaluated as follows:

$$\begin{aligned} E(S^x, O) &= 120 + max(0, 0 - 800) + max(0, 200 - 100) \\ &\quad + max(0, 0 - 100) + max(0, 0 - 0) \\ &= 120 + 0 + 100 + 0 + 0 \\ &= 220 \end{aligned}$$

The mechanism for extracting metric values from S^x is built upon the simulation detailed in the following section.

D. Build Order Simulator

The BO simulator is a program inspired by previous state-of-the-art implementations. It takes input parameters such as the input game state S^a, action space A, build order s, and provides the Genetic Algorithm (GA) with the output game state S^x. The output is described by solutions s in-game execution time and end state metrics S^{x^M} values. The simulator focuses only on the BO aspect of the SC2 game and does not simulate enemy players.

The program simulates every second of the game in a loop until every action in the build order is completed or timeout is reached. During the simulation, we execute actions, often concurrently. Once an action is executed, we add the provided resource to the game state. We extract metrics from the provided game state S^n for the E function when the simulation ends. The simulator loops through BO and tries to execute the current action in each time frame. If it cannot, it just skips to another time frame, so during the evolution process, we have to ensure that every build order in the population is valid.

The simulation ends with a timeout if the build order is invalid, meaning it is impossible to execute based on game rules. However, in the proposed implementation, timeout never occurs, and the simulator validates correct solutions.

Build order completion is not a deterministic process, but it can be approximated by one. For instance, the time to build a Pylon can vary based on the place we want to build it. Nevertheless, this time is often similar, so it could be approximated by a fixed amount of time. For every action that needs to significantly change the location of execution, such as creating a new Nexus base, additional time is needed for travel.

E. Genetic Algorithm for Planning and Scheduling (GAPS)

We attempt to solve Planning and Scheduling (PS) problem on the example of the StarCraft II Build Order Optimization (SC2 BO) problem using a Genetic Algorithm (see **Algorithm 1**). GA is advantageous for exploring expansive solution spaces and strategy games encapsulating complex, deterministic environments, providing an ideal ground for conducting research in this field. Therefore we name our method Genetic Algorithm for Planning and Scheduling (GAPS). We propose a straightforward solution generation method that creates valid length solutions n based on an action tree. In this tree, state S^a is the root. From there, one action is selected randomly from the list of possible options, which transitions the system to the next state. This process is repeated, updating the list of possible actions based on the actual state until the size of the created solution equals n. After initialization, each created solution is evaluated and sorted based on fitness. To evaluate build order, we employ function E as previously described. We search for optimal configuration of the rest of the operators like selection, crossover, mutation, and repair in experiments (Sec.IV).

In the context of GA, Lamarckianism gives that an individual can modify their genotype in response to their environment and subsequently pass this change on to their offspring. In contrast, the Baldwin Effect posits that individuals can make non-genetic (phenotypic) changes over their lifetime in response to their environment. Those that successfully adapt in this way are more likely to survive and reproduce. Over time, genetic changes supporting these phenotypic adaptations could become more prevalent in the population, leading to the genetic assimilation of learned traits. Applying Lamarckianism to a fixed-length Build Order (BO) is challenging, requiring solutions to be repaired without altering its length. We provide the pseudocode of our base repair function, which eliminates actions that cannot be executed (see **Algorithm 2**). We then incorporate the Lamarckian approach by filling in missing actions with a 'None' action and overriding the genotype.

We implement the Baldwin Effect by repairing and evaluating a copy of the genotype, leaving the original genotype unchanged in the population.

Algorithm 1 GAPS

Require: $gameState$, $actionSpace$, $solution$, $populationSize$, $generations$, $solutionSize$, $crossoverRate$, $mutationProb$

Initialize $population \leftarrow generatePopulation(...)$
for i = 0 **to** $generations - 1$ **do**
  $sort(population)$
  $best = population[0]$
  Initialize $children$ as an empty list
  $children.append(best)$
  for i = 0 **to** $populationSize - 1$ **do**
    $parent1$, $parent2$
    $selection(population, parent1, parent2)$;
    if $underProbabilityThreshold(crossoverRate)$ **then**
      $child \leftarrow crossover(parent1, parent2)$
    end if
    if $underProbabilityThreshold(mutationProb)$ **then**
      $child \leftarrow mutate(child)$
    end if
    $repairedSolution \leftarrow repair(child)$;
    $fitness \leftarrow evaluate(repairedSolution)$;
    $children.append(child)$
  end for
  $populaton \leftarrow children$
end for
return $population[0]$

Algorithm 2 Repair Algorithm

Require: $gameState$, $actionSpace$, $solution$

Initialize $resultBuildOrder$ as an empty list
for i = 0 **to** $solution.buildOrderSize - 1$ **do**
  $action \leftarrow solution.buildOrder[i]$
  if $isActionPossible(gameState, actionSpace, action)$ **then**
    $resultBuildOrder.append(action)$
  end if
end for
return $resultBuildOrder$

IV. Experiments

In this section, all research experiments are presented to answer four research questions:

- **RQ1** - How do different genetic operators (crossover, mutation) influence the evolution process? ($a30case$)
- **RQ2** - How does GA configuration (budget, selection, crossover rate, mutation probability) influence the evolution process? ($allcases$)
- **RQ3** - How effective is GA using Lamarckianism in comparison to Baldwin Effect? ($allcases$)
- **RQ4** - How effective selected GA configuration based on previous experiments is in comparison to some state-of-the-art setups in the context of all test cases? ($allcases$)

The following sections present details and results of developed experiments to answer the above research questions.

A. Test cases

We prepared three objective scenarios for enriching experiments: $aggressive$, $balanced$, and $development$. Each scenario is divided into three problem sizes: 30, 60, and 150, where problem size describes how long action sequences we want to examine. For 9 cases, we propose objective vectors $O = [m^1, m^2, m^3, m^4]$. In the $aggressive$ scenario, the goal is to gain as much army value as soon as possible, the development scenario aims to create a strong economy, and the balanced scenario is a mix of economy, army, research, and defense; we propose values of each O based on game experience. This approach allows us to examine the algorithm's behavior in response to varying types of problems in the context of different levels of complexity.

Based on our knowledge of the game and previous manual experiments, we propose setting budgets for all examined methods corresponding to three problem sizes. Additionally, we use a short code to name each case; for instance, an aggressive problem of size 30 is labeled $a30$. The case $a30$ will serve as the base case. For the $a30$ case, we define $O = [0, 2000, 0, 0]$, meaning that, given a maximum of 30 actions, a selected method needs to identify a build order that yields an army value of 2000 in the shortest possible makespan (see Tab.I on p.7).

For testing procedures in tuning/general experiments, as default *a30* scenario is used. For methods comparison, all test cases are used.

B. Setup

All experiments were done using the computer with AMD EPYC 7H12 64-Core Processor, Ubuntu operating system, and C++20. Because of the non-determinism nature of GA, for every configuration, we repeat the experiment 10 or 30 times, depending on the experiment.

For each case scenario, we propose to examine three different solution sizes described by $n = sum(O) * 1.5\%$. For example, in the case of $a30$ we examine population sizes 60, 90, and 150. We can define the number of generations as $number\,of\,generations$ = $budget/population\,size$. Therefore we have specific problem sizes for each scenario with defined budgets. For each problem size, we have three population size/generations proportions constrained by budget. For example, in the aggressive scenario for problem size 30, we have three population size/generations proportions: 60/750, 90/500, and 150/300.

We conduct crossover/mutation experiments incorporating three standard crossovers: one-point crossover, two-point crossover, and uniform crossover. We also examine three different mutations: random, bit flip, and mixed mutation presented in COEPc. For our configuration experiment, we validate three distinct population sizes/generations proportions within a specified budget across all nine scenarios, using

TABLE I
TEST CASES – SCENARIOS

Genotype size	**Budget**	**Aggressive**	**Balanced**	**Development**
30	45000	$a30$ $O = [0, 2000, 0, 0]$	$b30$ $O = [1100, 1000, 250, 150]$	$d30$ $O = [1500, 200, 0, 0]$
60	150000	$a60$ $O = [0, 4000, 0, 0]$	$b60$ $O = [2000, 1000, 400, 300]$	$d60$ $O = [2200, 400, 0, 0]$
150	750000	$a150$ $O = [0, 10000, 0, 0]$	$b150$ $O = [3300, 1000, 250, 150]$	$d150$ $O = [5500, 0, 0, 0]$

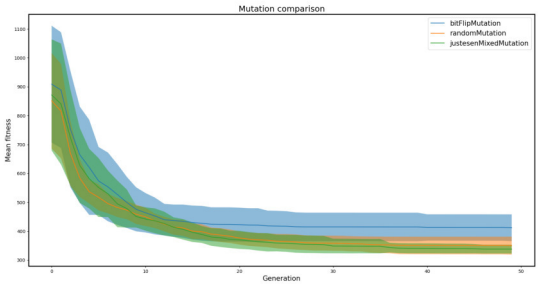

Fig. 2. Mutation comparison for $a30$

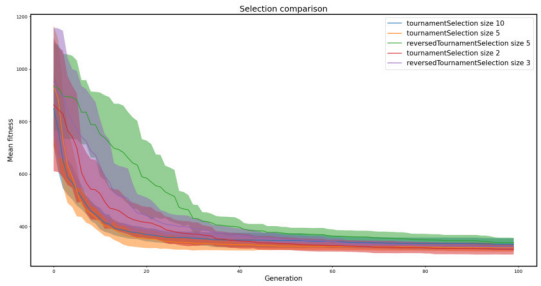

Fig. 3. Selection results for $a30$

TABLE II
MUTATION COMPARISON $a30$

Mutation type	mean	std_dev
Bit Flip	409.9	46.8
Mixed	314.3	19.2
Random	329.2	20.7

TABLE III
SELECTION COMPARISON TYPE $a30$ REPEATS

Tournament size	mean	std_dev
reversed 5	315.1	14.51
reversed 3	298.4	14.31
2	305.5	17.69
5	309.0	17.25
10	315.1	14.51

various tournament selection sizes, crossover rates, and mutation probabilities. It is important to note that mutation probability refers to the chance of applying a mutation that alters precisely one gene in the genotype, an approach inspired by state-of-the-art implementations. In addition to the standard configuration experiment, we investigate a further variant: a reverse tournament of size n that selects a parent randomly based on the $n-1$ best candidates and thus considerably lowers selection pressure.

Once we establish the GAPS configuration (GAPSc), we explore it in the context of Lamarckianism and the Baldwin Effect.

Finally, we compare GAPSc with BnB and MCTS as well as COEPc and MOEAc for a state-of-the-art comparison. State-of-the-art GA-based methods configurations:

- COEPc - random selection (best 25%), two-point crossover (crossover rate 100%), mixed mutation (probability 50%), Baldwin Effect-based repair,
- MOEAc - tournament selection (size 2), one-point crossover (crossover rate 90%), bit flip mutation (probability 100%), Baldwin Effect-based repair.

C. How does crossover/mutation influence GAPS effectiveness? – **RQ1**

The experiments with genetic operators (i.e. crossover and mutation) showed that although chosen crossover does not have much impact on the evolution, mutation can be quite vital (see Tab.II and Fig.2). The best mutation proved to be presented in COEP mixed mutation, which encapsulated four different mutations where each can be applied to created offspring. It helps enrich population exploitation of solution space, improving evolution quality.

D. How to configure the GAPS? – **RQ2**

In the initial stage of experiments, the configuration experiment proved the insignificance of validated population sizes.

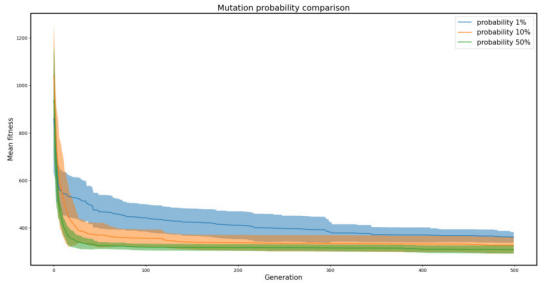

Fig. 4. Mutation probability comparison ($a30$)

TABLE IV
MUTATION PROBABILITY COMPARISON FOR $a30$, BEST FITNESS FOR 10 REPEATS

Mutation probability	mean	std_dev
1%	360.5	22.3
10%	327.5	35.1
50%	309.0	17.2

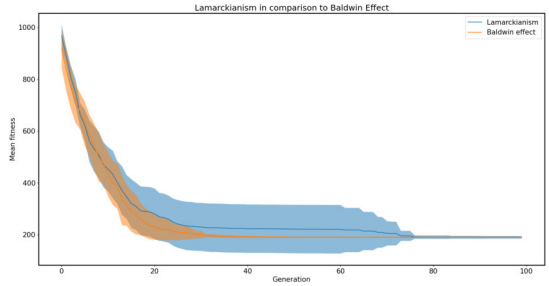

Fig. 5. Lamarckianism in comparison to Baldwin Effect for $d60$

TABLE VI
COMPARISON RESULTS OF BNB, MCTS AND GAPS FOR $a30$, BEST FITNESSES FROM 30 REPEATS

Method	mean	std_dev
GAPS	318.7	32.8
BnB	1933	*0.0*
MCTS	1827.3	43.1

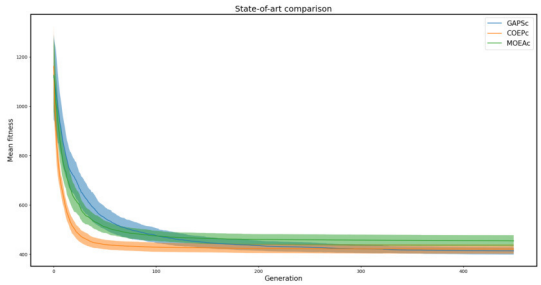

Fig. 6. Comparison results for GA for $b150$

It could be suggested that we search too few configurations to show a significant difference. However, this gives us valuable information about the evolution process.
The selection/crossover rate/mutation probability experiment provided us with the best configuration for the next experiments, which proved to be reversed tournament selection with size 3, crossover rate = 40%, and mutation probability = 50% – see Tab.III with Fig.3. and Tab.IV with Fig.4. Based on experimental results, we can define the best-found configuration of GAPS (GAPSc): random correct solutions generation, tournament selection (size 5), one-point crossover (crossover rate 30%), mixed mutation (probability 50%), and Baldwin Effect-based repair.

E. Lamarckianism in comparison to Baldwin Effect – **RQ3**

Results of GAPS in the context of Lamarckianism and the Baldwin Effect varies based on the scenario. At times, one outperforms the other. However, generally speaking, the Baldwin Effect tends to be the more stable choice (see Tab.V and Fig. 5).

F. State-of-the-art setups comparison – **RQ4**

Experimental results showed that BnB and MCTS methods could not find any solutions for any scenarios under a given budget because they cannot search for solutions of defined sizes (ex. 30). Therefore to extract some solutions, we compare them by giving the task of finding a solution with the most military units in a given budget and then comparing these solutions to the solution found by GA in $a30$ case (see Tab.VI). The experiment shows that under a budget of 45000, associated with a problem of size 30, BnB can find an optimal solution for acquiring 200 army value. MCTS, on the other hand, can find a suboptimal solution that can provide 300 army value. In comparison, for the same budget, GAPS is able to find a solution that achieves the given goal of 2000 army value on average in 318.7 in-game seconds.

The results from the GA-based experiments reveal interesting insights (see Table VII and Figure 6). These suggest that our GAPS configuration (GAPSc) often discovers superior solutions in smaller, $aggressive$ cases. Conversely, COEPc tends to generate better solutions in larger, more $balanced$ cases. The outcomes for $development$ cases are quite similar.

The explanation for this might be rooted in the specifications of the distinct scenario. $Aggressive$ cases typically require highly optimized solutions with a specific order of action execution. In contrast, $development$ cases permit a wider range of solutions, all yielding similar results.

TABLE V
LAMARCKIANISM IN COMPARISON TO BALDWIN EFFECT FOR $d60$, BEST FITNESSES FROM 30 REPEATS

Repair type	mean	std_dev
Lamarckianism	187.8	2.7
Baldwin Effect	188.9	1.9

G. Comparison to known build order

We conduct a final experiment in which we take a well-rated Stalker rush build order from a website that stores the best Sc2 build orders. We simulate this order, extract metrics, and then input those metrics as an objective vector for our implementation to check if GAPS can find a similar build order. The experiment reveals that while the website build order takes 221 in-game seconds, our method only requires 205 seconds to produce a build order described by the same objective vector. Although using the same actions, the solution provided by GAPS completely reorders them in a manner

TABLE VII
COMPARISON RESULTS FOR GA FOR ALL SCENARIOS

	GAPSc		**COEPc [2]**		**MOEAc [4]**	
Case	mean	std_dev	mean	std_dev	mean	std_dev
a30	**302.17**	11.34	306.83	20.27	356.63	37.29
b30	258.2	10.15	**251.26**	10.78	320.033	32.22
d30	**187.23**	2.06	**187.83**	2.29	207.03	39.16
a60	**409.3**	16.21	420.23	19.55	452.467	22.57
b60	318.5	14.5	**309.36**	7.99	373.36	27.17
d60	**246.57**	3.55	**245.2**	4.99	262.2	10.71
a150	731.0	9.1	**707.7**	20.25	785.9	20.96
b150	561.27	18.62	**534.56**	15.37	644.3	39.93
d150	**1422.13**	1.2	**1422.87**	1.83	**1422.83**	1.84

that allows a significant number of actions to be executed concurrently.

V. CONCLUSION

This study offers several key conclusions regarding applying and optimizing GAPS for planning and scheduling problems in the context of the game StarCraft II.

Mutation operators significantly impact the evolutionary process. The mixed mutation strategy described in [2] provided the best results, enhancing the population's ability to exploit the solution space and thereby improving the quality of evolution.

Search for optimal GAPS configuration, including population size, number of generations, selection, crossover rate, and mutation probability, reveals that population size and number of generations did not significantly impact the experiment results. The best-founded configuration proved to be: reversed tournament selection of size 3, one-point crossover with crossover probability of 40% as well as mixed mutation [2] with mutation probability of 50%.

The results were scenario-dependent when comparing Lamarckianism and the Baldwin Effect in GA. The Baldwin Effect emerged as a more stable choice across different problem sizes and scenarios. This stability is primarily due to the Baldwin Effect's ability to balance exploration and exploitation in the solution space, reducing the risk of premature convergence to suboptimal solutions.

In contrast to other state-of-the-art configurations, GAPSc demonstrated proficiency in smaller, aggressive cases where precision and optimization were vital. Conversely, COEPc prevailed in larger, balanced cases where flexibility and a range of solutions were advantageous. Therefore, careful consideration of case characteristics is crucial when deciding the most appropriate method to apply. In doing so, we can utilize the strengths of each method, ensuring optimal outcomes.

The final experiment underscored the potential of GA in optimizing game strategies. In comparison to a highly-rated Stalker rush build order from a renowned strategy website, the GA generated a similar build order in fewer in-game seconds, showcasing its potential to create efficient and competitive game strategies. The time difference (16sec.) might cause a significant difference at the very beginning of the high-ranked match.

In conclusion, the findings of this study underscore the effectiveness and potential of Genetic Algorithms in tackling complex planning and scheduling problems, similar to StarCraft II Build Order Optimization. Furthermore, they provide a foundation for future exploration and optimization in this domain.

VI. FUTURE WORKS

Several promising future research directions exist for enhancing the Genetic Algorithm applied to BO.

Firstly, the development of more sophisticated objective metrics could be beneficial. Such metrics would encapsulate the tactical nuances of each unit in the game, providing a richer representation of the problem domain within the evolutionary process. This enhancement could lead to solutions that better reflect the complex dynamics of the game.

A natural extension of this is the application of a multi-objective Genetic Algorithm. It would enable the simultaneous optimization of several objective functions, providing a more comprehensive optimization that considers the nature of planning and scheduling problems.

Using surrogate models, also known as meta-models, could help accelerate the evaluation process within the GA. These models, built based on existing evaluation data, can provide fast, approximate evaluations, significantly speeding up the evolutionary process.

Exploring different approaches could also increase the effectiveness. One promising method that aligns well with the nature of Genetic Algorithms in the context of presented results is Extreme Optimization (EO). This technique focuses on the most complex components of a solution and attempts to improve them. Thus, it could be particularly effective in complex environments like those encountered in StarCraft II, where optimizing numerous suboptimal solutions is necessary. Combining GA with EO could lead to a more efficient search process. While GA excels at exploring a broad solution space, adding EO allows it to focus more specifically on the areas that need the most improvement, leading to more exploitation of the solutions space and enhancing effectiveness.

Integrating GA with a high-level agent could validate its performance and potentially create a powerful StarCraft II bot. This agent could use the GA-generated solutions as part of its decision-making process, while the feedback from the agent's performance could further inform and refine the evolutionary process.

GAPS can be extended to numerous domains, such as economics, and military simulations, due to its inherent flexibility and adaptability. In the economic field, GAPS can be applied to optimize portfolio management, improve supply chain efficiency, and refine resource allocation plans in large-scale projects, effectively scheduling investments over time to maximize returns while minimizing risk. Similarly, in military simulations, GAPS can optimize strategies for defense and offense, logistics, troop deployment, and disaster response by assessing multiple scenarios accordingly.

The potential directions for enhancing the presented GA are vast and multidimensional, ranging from more nuanced objective functions to hybrid techniques and high-level agent integration. These could all contribute to a more robust, efficient, and capable GA for tackling the complex problem of game planning and scheduling.

Finally, more advanced simulations or games can be examined, to verify how the increased number of resource management options affects the results. It could include a higher number of resources, a resources trade/exchange system with a simple demand/supply mechanism, and soft constraints based on the simulation-specific rules. Those extensions can bring the environment significantly closer to real-world scenarios. Another interesting, yet challenging aspect to simulate is risk management. In both game and economic environments multiple players compete with each other, thus long-term and error-prone plans can be replaced by less efficient but safer strategies. It is related to the previously mentioned multi-objective optimization as the risk exposure might be one of the minimized objectives.

REFERENCES

[1] Kovarsky, A., and Buro, M. (2006) "A first look at build-order optimization in real-time strategy games." Proceedings of the GameOn Conference. 2006.

[2] Justesen, Niels and Risi, Sebastian. (2017). Continual online evolutionary planning for in-game build order adaptation in StarCraft. 187-194. 10.1145/3071178.3071210.

[3] Wei, LZ and LW Sun. (2009) “Build Order Optimisation For Real-time Strategy Game”, http://www.nus.edu.sg/nurop/2009/SoC/nurop-LimZhanWei.pdf

[4] Blackford, J., and Lamont, G. (2014) "The real-time strategy game multi-objective build order problem." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 10. No. 1. 2014.

[5] Churchill, D., and Buro M. (2011) "Build order optimization in starcraft." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 7. No. 1.

[6] Weber, Bryan S. (2018) "Standard economic models in nonstandard settings–starcraft: Brood war." 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE,.

[7] Buro, M., and Kovarsky, A. (2007) "Concurrent action execution with shared fluents.", AAAI Conf. 2007: 950-955.

[8] Fox, M., and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains." Journal of artificial intelligence research 20 (2003): 61-124.

[9] Vinyals, O., Babuschkin, I., Czarnecki, W.M. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354 (2019). https://doi.org/10.1038/s41586-019-1724-z

[10] Liu, Ruo-Ze and Pang, Zhen-Jia and Meng, Zhou-Yu and Wang, Wenhai and Yu, Yang and Lu, Tong. (2022) "On Efficient Reinforcement Learning for Full-length Game of StarCraft II", Journal of Artificial Intelligence Research 75, 2022, pp.213-260.

[11] El-Nabarawy, Islam and Arroyo, K. and Wunsch, D. (2020). "StarCraft II Build Order Optimization using Deep Reinforcement Learning and Monte-Carlo Tree Search", https://arxiv.org/pdf/2006.10525.pdf.

[12] M. Kuchem, M. Preuss and G. R. (2013) "Multi-objective assessment of pre-optimized build orders exemplified for StarCraft 2" 2013 IEEE Conference on Computational Intelligence in Games (CIG), 2013, pp. 1-8, doi: 10.1109/CIG.2013.6633626.

[13] C.W. Leung, T.N. Wong, K.L. Mak, R.Y.K. Fung, (2010) Integrated process planning and scheduling by an agent-based ant colony optimization, Computers and Industrial Engineering, Vol. 59 (1), pp.166-180.

[14] Cobb, Charles W., and Paul H. Douglas. "A theory of production." (1928).