Data-driven Production Constrained Build-order Optimization in *StarCraft*

Pengpeng Wang[1], Yifeng Zeng[3] Langcai Cao[2]

1. Department of Automation, School of Aerospace Engineering, Xiamen University, Xiamen 361102, P. R. China(e-mail: ppwang@stu.xmu.edu.cn)

2. Department of Automation, School of Aerospace Engineering, Xiamen University, Xiamen 361102, P. R. China(e-mail: langcai@xmu.edu.cn)

3. School of Computing, Teesside University, Middlesbrough, TS1 3BX, U.K.(e-mail: Y.Zeng@tees.ac.uk)

Abstract: Production constrained build-order optimization problems challenge artificial intelligence research in computer game applications due to an uncertain set of constraints. Traditional approaches provide subjective values in the constraint formulation therefore resulting in unexpected performance of the optimal build-order in a game. In this article, we propose a data-driven approach to solve a build-order optimization problem in *StarCraft*. We formulate the constraint by learning the parameter values from game replay data, which complements more precise problem formulation. To solve the optimization, we use the improved genetic algorithm by learning initial solutions from the data. We show the performance of the data-driven methods in a *StarCraft* simulation platform.

Key Words: Real-time Strategy Games, Data-Driven Optimization, StarCraft, Genetic Algorithm

1 Introduction

Computer games have become an important and active area in artificial intelligence (AI) research since they not only provide a popular platform for technological evaluation but also elicit a new line of research e.g. game AI, therefore driving the AI research development. *StarCraft* is a well-known real-time strategy game released by *Blizzard Entertainment* [1]. and has engaged with AI research over years [1–3] that includes AI planning [4–6], intelligent agent control[7–9], prediction[10–12] and so on. There has seen a growing line of research on build-order optimization that is often required in the early stage of the *StarCraft* game. Given a specific build-order plan, players take actions to produce different types and numbers of units, buildings, upgrade technology or conduct other operations so as to obtain a dominant position in the early phase of a game. In this article, we re-visit the build-order optimization problem and develop a new solution in *StarCraft*.

A good build-order enables players to maximize the number of buildings and units in the production process where units differ in skills that are subsequently needed to construct different buildings. Hence the build-order optimization problem essentially involves a set of constraints that are naturally related to many factors in an uncertain gameplay environment. It is often referred as a *Production Constrained Build-order Optimization Problem* (PC-BO) in *StarCraft*. For example, one of the most important constraints is about formulation of the number of units that are needed to construct and subsequently protect the buildings. However, it is difficult to set a precise constraint in the optimization problem since it is also relevant to the number of units produced by players' opponents in the gameplay. Without formulating the proper constraint solutions to PC-BO often lead to unacceptable performance in the build-order planning.

To address the challenge of formulating a precise PC-BO, we resort to a data-driven approach to the optimization problem that exploits game replay files to facilitate the unit calculation in *StarCraft*. The approach is motivated by widely available replay files that are uploaded by various game players in the world. The replay files record players' interactions in the gameplay e.g. the building times, the unit production of both players and their opponents and the types of the units. Subsequently we adapt the traditional genetic algorithms to solve the optimization problem with data-driven constraints. To further improve the algorithm performance, we learn an initial solution of a build-order instead of randomly generating the solution in the algorithm development.

The rest of this paper is organized as follows. We introduce background knowledge and related works in Section 2. In Section 3, we present the mathematical model of PC-BO and particularly elaborate the constraint development. We adapt the genetic algorithm to solve PC-BO in Section 4 and show the experimental results in Section 5. Section 6 concludes this paper.

2 Background and Related Works

2.1 StarCraft

In *StarCraft*, tasks can be categorized into two types of managements namely "micro-management" and "macro-management" [6]. Controlling units to move, attack or evade from enemies is related to micro-managements; while, behaviours like constructing buildings, producing units and scouting enemies are macro-managements. Macro-management is considered as a high-level strategy making process. The problem we consider in this paper belongs to macro-management which is to make a build-order in order to produce buildings, soldiers, defensive buildings/uints or other special units in the early game stage.

In *StarCraft*, a player can choose one of three races each of which has different buildings and armies. At the beginning of the game, the player has a base, several workers and minerals and can control workers to gather resources or

This work is supported by National Natural Science Foundation (NNSF) of China under Grant 61836005.

build buildings. Producing buildings and units consume resources while units can only be produced in the buildings and some buildings can't be constructed without constructing other specific buildings. The player needs to develop a reasonable build-order for the purpose of defeating his opponents. After having different types and numbers of units, the player will conduct a real-time control of the units such as attack, defense and reconnaissance. Due to the fog of war, the player can only gather incomplete information about game states and can not see the opponents' states without scouting the game environments, which challenges AI research [1].

2.2 Build-order Optimization

Build-order optimization is to find a build-order with the objective of maximizing the production of units and buildings given relevant constraints. Churchill *et al.* [5] applied a depth-first branch and bound for searching a build-order that minimizes the time used to reach a given goal. Macret *et al.* [13] used a genetic algorithm to maximize the number of produced objects in a given time period or to produce a certain number of objects as fast as possible. Multi-objective evolutionary algorithms have also been used to optimize a build-order in multi-objective problems [14–16]. Justesen *et al.* [6] proposed continual online evolutionary planning techniques to evolve build-orders continually according to game states during the game. However, the aforementioned approaches do not involve a direct use of gameplay data and depend on inputs solely from game experts.

2.3 Data-driven Optimization

It is always a difficult task to set up a precise mathematical model in an optimization problem since the model parameters are full of uncertainty and it is not easy to specify parameter values [17–20]. Meanwhile, establishing a precise mathematical model is computationally expensive [21–24]. Hence much research on optimization has developed a data-driven approach that exploits problem domain data to build the optimization model and subsequently solve the model [21]. The approach namely data-driven optimization is strongly motivated by currently available data in various applications. Particularly in order to improve the robustness of optimization under uncertainty, existing research focuses on designing an uncertain set. Wang *et al.* [17] defined an accessible distribution set to contain only those distributions that make the observed data achieve a certain level of likelihood and apply their method to a portfolio selection problem. Bertsimas *et al.* [25] proposed a data utilization schema so as to design uncertainty sets for a robust optimization, which is used in portfolio management and queueing. Mevissen *et al.* [26] employed polynomial and histogram density estimates to approximate the distribution of uncertain parameters and applied their model and solutions in a water network problem. Wang *et al.* [21] used a surrogate-assisted multi-objective evolutionary algorithm to solve the optimal configuration problem of trauma systems. The technique relies on the data and is computationally expensive. By using a regression and logarithmic transformation, Yang *et al.* [24] transformed a nonlinear mixed integer problem that contains queueing model into an integer linear problem, and solved the transformed model in an efficient way.

Due to the fog of war in *StarCraft*, players can't observe opponent's states directly in gameplay. Consequently, we can't precisely specify the number of armies for the constraint in a PC-BO problem. In this paper, we learn the opponents' states from available game replay data and solve a PC-BO problem through a data-driven approach.

3 Production Constrained Build-order Optimization

3.1 Problem Description

In a PC-BO problem, we aim to maximize the number of units in the early stage of *StarCraft* that can be divided into several time periods from a game start to a certain gameplay point. One natural thought is to maximize the number of units for every time period thereby leading to the largest sum of units in the PC-BO problem. However, this is not entirely correct since there is a main constraint namely *Technology Tree* in *StarCraft*. More concretely, different races have different units in *StarCraft* and every unit has different abilities or skills. For example, the *Siege Tank* has strong firepower and armor which is a kind of powerful ground force, but it can't attack flight targets. Moreover, to produce different units need different amounts of resources. Some units need to be produced in building that require other types of buildings. Hence a sequence of buildings need to produced in order to successfully generate the units. For example, without *Barracks* we can't produce *Marine*. The relations between units and buildings shall be considered as a main constraint in a PC-BO problem.

Meanwhile, the quantity of units needs to be considered in the aforementioned relations since some units can't be produced without sufficient amount of other units. The setting of unit quantity is extremely hard since it also depends on the number of units to be produced by other players. In general we let the number of units to exceed a threshold of unit quantity that needs to be estimated at every time period. In this paper, we particularly consider it as one quantity constraint and propose a data-driven approach to learn a good estimation in a PC-BO problem.

3.2 PC-BO Formulation

We present the objective function of a PC-BO problem in Eq. 1.

$$\max \sum_{t}^{T} \sum_{i}^{U} Unit_{t,i} \tag{1}$$

where $Unit_{t,i}$ is the number of unit i to be produced at the time period t. T is the number of time periods that compose the early game stage while U is the number of unit types in *StarCraft*.

The constraints of PC-BO are:

$$GameConstraints \tag{2}$$

and

$$\begin{aligned} &Unit_{t,i} \geq \xi, \\ &t = 1, 2, \ldots, T; i = 1, 2, \ldots, U \end{aligned} \tag{3}$$

$GameConstraints$ include *supply constraints*, *technology constraints*, *resource constraints* and *time constraints* [14, 15, 27].

- Supply constraints: units require supply spaces. and new units won't be produced if current available supply spaces are less than need.
- Technology constraints: some buildings/units can't be built without other relevant buildings/units. For example, without existing Barracks, we can't construct Factory in *StarCraft*.
- Resource constraints: producing a building/unit needs a sufficient supply of other resources e.g. mineral, gas and so on.
- Time constraints: producing a building/unit needs a certain amount of times.

The above mentioned $GameConstraints$ can be encoded according to a technology map in *StarCraft*. Meanwhile, a PC-BO problem needs to consider the production quantity constraint in Eg. 3. The constraint requires to specify a numerical value, e.g. the value of ξ in Eg. 3, which leads to the difficulty of formulating a precise optimization problem. To address this difficulty, we first introduce penalty function below to reformulate the objective function and then learn the value of ξ from the data.

Definition 1 (Penalty Function) Define function:

$$\delta(x-\omega) = \begin{cases} x-\omega & x \geq \omega \\ x-\omega-N & x < \omega \end{cases} \tag{4}$$

Then, the objective function can be rewritten as:

$$\max \sum_{t}^{T} \sum_{i}^{U} \left(Unit_{t,i} + \delta\left(Unit_{t,i} - \xi\right)\right) \tag{5}$$

The parameter N is used to penalise the insufficient unit number, which will be set as the complexity of solutions to the PC-BO problem.

3.3 Value Learning for Production Constraints

The production quantity constraint needs to specify the value of ξ that is mainly depends on behaviour of opponents in a game particularly the number of units of different types used by the opponents in an attack. In principle, we can set the ξ value as the numbers of units sent by the opponents in one time-period. However, it is rather hard to get the precise number of opponents' units due to the fog of war. To address this difficulty, we will use the number of the units that can bee seen by players and historical data on the units used by their opponents.

We consider the unit production quantity as a multiple random variable without knowing its true distribution function or probability density function. We use *game replay files* as the historical data to learn the distribution units to be sent by opponents in a time-period. Below we give definitions to facilitate the constraint learning.

Definition 2 (Complete Domination) *Given the vectors $\alpha \in R^n$, $\beta \in R^n$, for each $i = 1, 2, \cdots, N$ and exist $\alpha_i > \beta_i$, we said $\alpha > \beta$ or α completely dominates β.*

Definition 3 (Domination) *Given the two vectors $\mathcal{A} \in R^{n\cdot m}$, $\mathcal{B} \in R^{n\cdot m}$ and*

$$a \in R^n = \left(\sum_{i=1}^{m} A_i, \sum_{i=m+1}^{2m} A_i, \quad \ldots, \sum_{i=(n-1)\cdot m+1}^{n\cdot m} A_i \right)^T,$$

$$b \in R^n = \left(\sum_{i=1}^{m} B_i, \sum_{i=m+1}^{2m} B_i, \quad \ldots, \sum_{i=(n-1)\cdot m+1}^{n\cdot m} B_i \right)^T$$

If $A > B$(Definition 1) , else if $a > b$ (Definition 1) , else if $\sum_{i=1}^{n\cdot m} A_i > \sum_{i=1}^{n\cdot m} B_i$, we say that A dominate B. If A does't dominate B and B doesn't dominate A, we say that the rank of A and B is equal.

According to Def. 2, the ranks of A and B are equal if and only if $\sum_{i=1}^{n\cdot m} A_i = \sum_{i=1}^{n\cdot m} B_i$. This is because if A doesn't dominate B, we have $\sum_{i=1}^{n\cdot m} A_i \leq \sum_{i=1}^{n\cdot m} B_i$ and if B doesn't dominate A either, we have $\sum_{i=1}^{n\cdot m} B_i \leq \sum_{i=1}^{n\cdot m} A_i$. Hence $\sum_{i=1}^{n\cdot m} A_i$ must be equal to $\sum_{i=1}^{n\cdot m} B_i$.

We may interpret Def. 2 in the context of *StarCraft* games. If there are m kinds of units waiting for production and the early game stage is divided into n time-periods and the opponent units-vector discovered by the player P_1 was A, the opponent units-vector discovered by the other player P_2 is B. $A > B$ indicates that the opponent units discovered by P_1 are larger than those identified by P_2 during every time-period. $\alpha > \beta$ means that the number of the opponent units discovered by P_1 are larger than P_2 for every time-period; but not every unit is greater than P_2. $\sum_{i=1}^{n\cdot m} A_i > \sum_{i=1}^{n\cdot m} B_i$ means that the total number of opponent units discovered by P_1 is larger than what P_2 find in the early game stage.

Theorem 1 *If A dominate B and B dominate C, then A dominate C.*

Proof. Proof by *contradiction*. There exist two cases: either the ranks of A and C are equals or C dominates A given that A doesn't dominate C. If A and C have identical ranks, we have $\sum_{i=1}^{n\cdot m} A_i = \sum_{i=1}^{n\cdot m} C_i$. However, this is contradicted by the fact that A dominates B, B dominates C by the reason that if A dominates B, we have $\sum_{i=1}^{n\cdot m} A_i > \sum_{i=1}^{n\cdot m} B_i$ and B dominates C, we have $\sum_{i=1}^{n\cdot m} B_i > \sum_{i=1}^{n\cdot m} C_i$, then we get $\sum_{i=1}^{n\cdot m} A_i > \sum_{i=1}^{n\cdot m} C_i$. Otherwise if C dominates A, we have $\sum_{i=1}^{n\cdot m} C_i > \sum_{i=1}^{n\cdot m} A_i$ which is also contradicted to the fact that A dominates B, B dominates C. Hence, the theorem holds. □

As a result of Def. 2, we can sort units-observed vectors of different players and get the unique sorting result. More specifically, we can sum the elements in different units-vectors and sort the results.

The sorting result can be used as the data distribution denoted by $\Pr(x)$. When the number of units-vectors dominated by r is k and the total number of units-vectors are N, we have $\Pr(x < r) = \frac{k}{N}$. Given a value $p \in [0, 1]$, the constraints parameterized by the random variable ξ can be obtained from $\Pr(Units\_Observed \leq \xi) \geq p$ where the $Units\_Observed$ is the units-vector. As a good build-order expects to generate at least ξ, e.g. $f(Build\_Order) \geq \xi$ where $f(Build\_Order)$ is a function that generates the number of units we can get in every time-period if we run the *build-order* in $StarCraft$. Subsequently, we have

$$\Pr\left(Units\_Observed \leq f\left(Build\_Order\right)\right) \geq p \qquad (6)$$

We use the data from professional player's replays. By sorting the observed-units-vectors from the replay data, we treat the sorting result as one-dimensional distribution. Given a value of p, the $k^{th} = (N \cdot p)^{th}$ observed-unit-vectors is used as the value of ξ. If there are more than one unit-vector in the k^{th} position, we take their average as the value of ξ. Examples of the specific value of ξ will be showed in the experiments.

4 Improved Genetic Algorithms for Solving PC-BO

We adopt a genetic algorithm [28] to solve the build-order optimisation problem. The genetic algorithm initialises a population of individuals, and each individual called a chromosome is a solution to the optimisation problem and is improved in an iterative way. The individual is evaluated by the objective function and the evaluation result is a fitness value. Using fitness values to select individuals to form a new population. Additionally, crossover and mutation operations are used to produce offsprings and compose the new population. The algorithms continues this iterative process until a termination condition is satisfied.

As the constraints in a PC-BO problem make the solution region discrete and the connectivity poor, it is difficult to obtain a satisfied solution within a certain number of generations through a simple genetic algorithm. We use simulated annealing [29, 30] to improve a local search in the genetic algorithm [31–33].

Meanwhile, it is noticed that initialising a good population is important and may improve the solution quality by providing a good solution point to the genetic algorithm. Given game replay data, we can learn a set of good examples of players' build-orders that lead to successful game outcomes and subsequently we use the learned build-orders to initialise the population. The improved performance will be verified in the experiments.

Running the genetic algorithm to solve the PC-BO problem demands to evaluate a large number of chromosomes each of which represents a potential build-order solution. The evaluations is conducted in every iteration and would cost a significant amount of computational times. To improve the evaluation efficiency, we develop a *StarCraft* simulator to run all the potential build-orders. The simulator simplifies the resource collection and building construction process.

Algorithm 1 elaborates the main process. The algorithmic improvements lie in the good initilization of populations for build-order solutions through game replay data (Step 6) and the introduction of simulated annealing process in the local search (Step 16).

Algorithm 1: Data-Driven Simulated Annealing Genetic Algorithm(Data_DrivenSAGA)

```
Input: parameters for genetic algorithm; parameters for simulated annealing; SortedPlayersBuildOrders; ProductionConstraints; TopK
Output: champion_chromosome; bestfitness
 1: //TopK is the number of players' build-orders in the top k.
 2: //The algorithm will stop either the Max_Generation is reached or the temperature t is low enough.
 3: t = $T_0$;
 4: pop = $\emptyset$;
 5: for i = 1 to TopK do
 6:     pop.add(SortedPlayersBuildOrders(i));
 7: end for
 8: for i = TopK + 1 to PopSize do
 9:     buildorder = randomly generated buildorder;
10:     buildorder = adjust(buildorder);
11:     buildorder.UnitsObtain = StarCraftSimulator(buildorder);
12:     buildorder.fitness = GetFitness(buildorder.UnitsObtain, ProductionConstraints);
13:     pop.add(buildorder);
14: end for
15: for i = 1 to Max_Generation do
16:     newpop = Simulated_Annealing(pop, t);
17:     newpop1 = Selection(newpop);
18:     newpop2 = CrossOver(newpop1, pCrossover);
19:     offspring = Mutation(newpop2, pMutation);
20:     fValueGet(offspring);
21:     pop = offspring;
22:     champion_chromosome = the best build order;
23:     bestfitness = champion_chromosome.fitness;
24:     //temperature falls;
25:     t = $\alpha \cdot t$;
26:     if t $\leq \varepsilon$ then
27:         break;
28:     end if
29: end for
30: return champion_chromosome, bestfitness;
```

5 Experiments

We formulate the PC-BO problem and implement the improved genetic algorithm to solve the optimization problem through exploiting the game replay data in *StarCraft*. We develop a *StarCraft* simulation platform and compare our data-driven approaches to the traditional approach that does not use any replay data.

5.1 Game Replay Data and Experimental Settings

The data we used is gathered by Gabriel Synnaeve[1] [34]. It contains 7659 *1 v.s. 1* replays of professional players. There are total 462 replays of *Terran v.s. Terran*. We remove 3 corrupted replays and keep a total of 459 replays in our experiments.

We ran the replays in *StarCaft* by using *Brood War API* [2]. We recorded the observed non-repetitive units every 12 frames (every unit has a unique ID). Other events like creation, destruction, change of ownership for each unit are also maintained. The information is saved into files with the **.rdata* format.

Subsequently, we can obtain the units' quantity observed by players from these the gathered files. We set the number of time-periods (denoted by T) as 4 and every time-period lasts for 3 minutes. We use 12 types of units in the experiments namely *Firebat*, *Ghost*, *Goliath*, *Marine*, *Medic*, *Siege_Tank*, *Terran_Vulture*, *BattleCruiser*, *Dropship*, *Science_Vessel*, *Wraith* and *Valkyrie*. We didn't consider *SCV* and *Vulture_Spider_Mine* because we regard the *SCV* as a

[1] http://emotion.inrialpes.fr/people/synnaeve/TLGGICCUP_gosu_reps.7z
[2] BWAPI https://github.com/bwapi/bwapi

worker and the *Vulture_Spider_Mine* as a mine rather than a fire-unit.

Given that the value of p equals to 0.5, we can identify six types of units that do not have all quantities of zeroes over all time-periods as shown in Table 1. Thus the objective function has $U = 6$.

Values in Table 1 are cumulative values. The reason for doing this is that units will not disappear until it is destroyed. For example, if we discover one *Marine* in the first time-period and discover two *Marine* in the second time-period, the value in the second row of *Marine* is set as 3. At the same time, due to the varying gameplay lengths, the value of units in the last row is filled with the value from the previous row if the game time is fewer than 12 minutes. For example, if the game ends in the eighth minutes, the value of the fourth row is filled with the third row. Because the unit production quantity is an integer, we rounded off values in Table 1 and obtained the approximate values in Table 2 that were used in formualting production constraints.

In addition, we extracted potential build-orders from the maintained files in the order of player's actions and skipped those build types that are not included in Table 3. From the replay files, we retrieve a number of build-orders given the number of builds in the experiments. We get 799, 715 and 624 build-orders for 100, 150 and 200 builds respectively.

5.2 Experimental Results

We ran all experiments through MATLAB R2016b on an Intel(R) Core(TM) i5-4590 CPU at 3.3GHz running Windows 7 with 8 GB of DDR3 1600MHz RAM. All algorithms were run in a single thread. We compare the traditional approach without using players' build-orders, i.e. the initial solutions of GA are randomly generated , to our approach as developed in Sections 3-4.

We set the parameters of our approach 1 as follows: *pCrossover* = *0.9*, *pMutation* = *0.05*, *initial temperature*(T_0)= *1000*, *temperature drop ratio*(α) = *0.9*, *minimum temperature*(ε) = *0.0001* and *Max_Generation* = *200*. This leads to the actual generations of 153 when running Algorithm 1. If not specified, the parameter settings are also used in other experiments. In the experiments, we set the number of builds as 100, 150 or 200, and the population size is set to either 150 or 200. Thus there are 6 combinations of parameters.

Fig. 1 shows the best fitness values of each generation averaged over 30 runs of Algorithm 1. For example, $L100P150$ is the result of the optimal solution with the chromosome length of 100 and the population size of 150 averaged over 30 independent runs.

Table 4 shows the best fitness values and their standard deviations, as well as the average runtime of Algorithm 1. It can be found that given the same chromosome length, a larger size of population provides better solutions while increasing the running time. In Fig.1, when the chromosome length is 200, the average fitness of the solutions is smaller than the solutions for other chromosome lengths regardless of the population size of 150 or 200. This should be due to an insufficient number of generations for the increasing length of chromosome.

When using players' build-orders from game replay file,

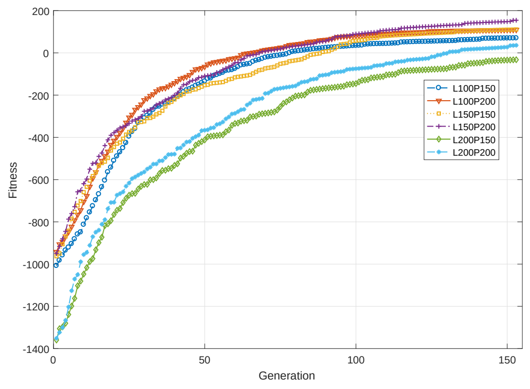


Fig. 1: The average fitness values over generations for the genetic algorithm without using players' build-orders.

we ran them in *Starcraft* simulator and then used the objective function to calculate the fitness. We then sort the build-orders in the descending order of their fitness values. The population size is set to 200 and chromosome length is set to 100, 150 and 200 respectively. We use the $TopK$ = 1, 5, 20, 50, 100 and 200 numbers of players' build-orders as the initial solutions for the genetic algorithm and other initial solutions are randomly generated if it is applicable. We show experimental results in Fig. 2, Fig. 3 and Table 5. All of results are averaged over 30 experiments.

We can see that the averaged solutions obtained by using players' build-orders of different sizes are better than the results without using players' build-orders. The variance is smaller as well under the same parameters. At the same time, we can find that when the chromosome length is 200, we get a better solution by using the players' build-orders as the initial solutions for Algorithm 1 without increasing the evolutionary generations. In Table 5, the maximum fitness from the results of using the players' build-orders is not inferior to the results without using players' build-orders. However, the use of replay data increases the runtimes of Algorithm 1; however, the increasing is not a bit. In addition, there is no much difference among the approaches using the players' build-orders of different sizes. In general, we get better results when 1/4 initial solutions of players' build-orders are generated from game replay data and the other 3/4 initial solutions are randomly generated.

6 Conclusions

In this paper, we focus on the production constrained build-order optimization (PC-BO) problem in $Starcraft$. Because the production constraint involves an uncertain production quantity, we use the player's game replay data to formulate a reasonably good optimization problem and solve the problem by using a genetic algorithm. By learning the player's build-orders from the game data, we use them as the initial solutions in the genetic algorithm. Through the $StarCraft$ simulation platform, we demonstrate that our new approach is better than the traditional technique that depends on the input of domain experts.

Although this work is focus on the build-order planning in *Terran v.s. Terran (TvT)*, our method can be converted to the build-order planning for other races. For example,

		Units					
		Marine	Goliath	Siege Tank	Vulture	Dropship	Wraith
Time-period	1	0.57	0.00	0.00	0.00	0.00	0.00
	2	2.65	0.26	1.17	1.52	0.00	0.43
	3	2.83	2.52	4.00	1.83	0.30	1.65
	4	2.83	6.57	12.43	2.78	1.39	2.26

Table 1: By using the method in Section 3 and setting p as 0.5, we get the quantities of observed units from game replay data. We exclude the units that are all zeroes over the four time-periods.

		Units					
		Marine	Goliath	Siege Tank	Vulture	Dropship	Wraith
Time-Period	1	1	0	0	0	0	0
	2	3	0	1	2	0	0
	3	3	3	4	2	0	2
	4	3	7	12	3	1	2

Table 2: The values come from rounding off the data in Table 1. We use these values in solving PC-BO.

Supply_Depot	Refinery	Barrack	Engineering_Bay	Academy
1	2	3	4	5
Factory	Machine_Shop	Starport	Armory	Control_Tower
6	7	8	9	10
SCV	Marine	Goliath	Siege_Tank	Vulture
11	12	13	14	15
Dropship	Wraith	Command_Center		
16	17	18		

Table 3: Codes of buildtypes for genetic algorithm.

number of builds	*PopSize*	*BestFitness*	*Time*(t)
100	150	69.87 ± 109.55	89.42 ± 1.62
150	150	111.07 ± 143.93	120.55 ± 3.22
200	150	-32.33 ± 234.42	142.22 ± 4.43
100	200	109.27 ± 88.79	119.61 ± 1.80
150	200	154.40 ± 104.94	160.99 ± 4.07
200	200	35.87 ± 182.02	193.69 ± 6.13

Table 4: The results of without using players' build-orders over 30 independent runs. *number of builds* with the same meaning of chromosome length.

the production constraints can be handled by adding the unit matchup information between different races [6]. Similar to development of genetic algorithm based approach, our approach still requires the exploration of parameter settings in the experiments. We are planning to conduct future research on automating the parameter settings by further exploiting domain data.

References

[1] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in Starcraft," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013.

[2] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, (2017). "Starcraft II: A new challenge for reinforcement learning," pp. 1–20. [Online]. Available: https://arxiv.org/abs/1708.04782

[3] G. Robertson and I. Watson, "A review of real-time strategy game AI," *AI Mag.*, vol. 35, no. 4, pp. 75–104, 2014.

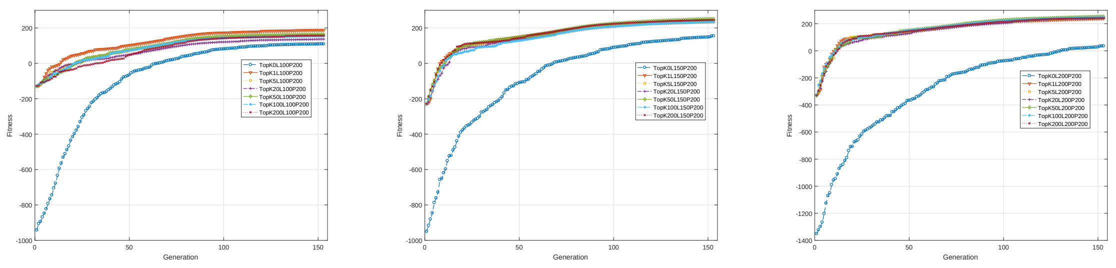
[4] B. G. Weber and M. Mateas, "Case-based reasoning for build order in real-time strategy games," in *Proc. 5th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, 2009, pp. 106–111.

[5] D. Churchill and M. Buro, "Build order optimization in Starcraft," in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2011, pp. 14–19.

[6] N. Justesen and S. Risi, "Continual online evolutionary planning for in-game build order adaptation in Starcraft," in *Proc. Genetic Evol. Comput. Conf.*, 2017, pp. 187–194.

[7] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to Starcraft micromanagement tasks," in *Proc. Int. Conf. Learning Representations*, 2017.

[8] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game Starcraft:broodwar," in *Proc. IEEE Conf. Comput. Intell. Games*,



(a) Results of all generations when the chromosome is 100.

(b) Results of all generations when the chromosome is 150.

(c) Results of all generations when the chromosome is 200.

Fig. 2: The averaged fitness values over generations for the genetic algorithm using different sizes of players' build-orders and different lengths of chromosome. The size of population used here is 200. $TopK0L100P200$ denotes the algorithm with the setting of 100 chromosomes without using players' build-orders from replay data (TopK0); while, $TopK1L100P200$ uses one build-order (TopK1) and 100 chromosomes.

TopK	*number of builds*	*BestFitness*	*MaxFitness*	$Time\ (s)$
0	100	109.27 ± 88.79	219	119.61 ± 1.80
0	150	154.40 ± 104.94	273	160.99 ± 4.07
0	200	35.87 ± 182.02	291	193.69 ± 6.13
1	100	187.67 ± 30.50	215	131.36 ± 1.19
1	150	241.40 ± 17.51	279	174.00 ± 2.65
1	200	239.67 ± 21.16	271	180.79 ± 4.28
5	100	158.47 ± 75.66	227	129.45 ± 1.73
5	150	234.00 ± 26.11	265	173.44 ± 1.80
5	200	242.13 ± 17.18	279	182.61 ± 3.13
20	100	136.73 ± 81.77	215	129.48 ± 1.18
20	150	232.80 ± 53.36	273	172.70 ± 2.18
20	200	239.47 ± 26.22	305	182.92 ± 3.88
50	100	163.73 ± 65.11	215	128.44 ± 1.10
50	150	250.73 ± 18.69	283	171.34 ± 1.99
50	200	252.60 ± 29.34	313	183.75 ± 4.01
100	100	154.67 ± 60.93	225	128.06 ± 1.07
100	150	234.27 ± 54.84	281	171.16 ± 1.67
100	200	247.80 ± 22.16	299	182.66 ± 3.53
200	100	156.73 ± 69.08	217	127.60 ± 1.30
200	150	244.93 ± 18.21	303	171.12 ± 1.84
200	200	242.13 ± 46.79	315	180.30 ± 4.03

Table 5: The results of Algorithm 1 by using different size of players' build-orders over 30 independent runs. *MaxFitness* is the maximum fitness we got among these 30 experiments.

2012, pp. 402–408.

[9] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in Starcraft," in *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, ON, Canada, 2013, pp. 1–8.

[10] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Proc. Int. Symp. Comput. Intell. Games*, 2009, pp. 140–147.

[11] H. C. Cho, K. J. Kim, and S. B. Cho, "Replay-based strategy prediction and build order adaptation for Starcraft AI bots," in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 1–7.

[12] H.-S. Park, H.-C. Cho, K. Lee, and K.-J. Kim, "Prediction of early stage opponents strategy for Starcraft AI using scouting and machine learning," in *Proc. of the Workshop at SIGGRAPH Asia, ACM*, pp. 7-12, Nov. 2012.

[13] M. Macret, (2010). "Build-order optimization in real-time strategy games using evolutionary techniques," pp. 1–7. [Online]. Available: http://matthieumacret.com/pdf/IAT813_MACRET.pdf

[14] H. Köstler and B. Gmeiner, "A multi-objective genetic algorithm for build order optimization in Starcraft II," *KI - Künstliche Intelligenz*, vol. 27, no. 3, pp. 221–233, 2013.

[15] J. Blackford and G. Lamont, "The real-time strategy game multi-objective build order problem," in *AIIDE*, 2014.

[16] M. Kuchem, M. Preuss, and G. Rudolph, "Multi-objective assessment of pre-optimized build orders exemplified for Starcraft 2," in *Proc. IEEE Comput. Intell. Games*, 2013, pp. 1–8.

[17] Z. Wang, P. W. Glynn, and Y. Ye, "Likelihood robust optimization for data-driven problems," *Comput. Manag. Sci.*, vol. 13, no. 2, pp. 241–261, 2016.

[18] B. J. Eck and M. Mevissen, "Valve placement in water networks: Mixed-integer non-linear optimization with quadratic pipe friction," *Report No RC25307 (IRE1209-014), IBM Research (September)*, 2012.

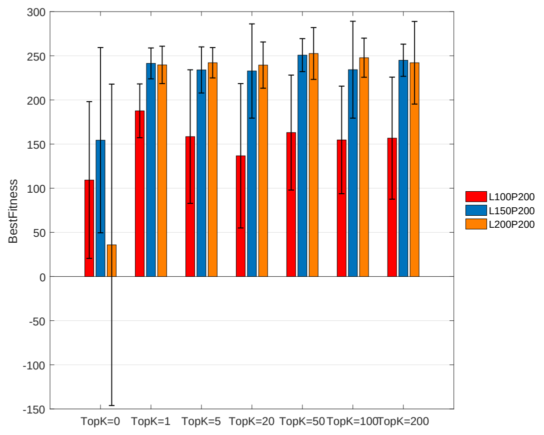


Fig. 3: The best fitness values and their standard deviations over 30 independent experiments when using different size of population and different lengths of chromosome.

[19] R. Zhang and J. Tao, "Data-driven modeling using improved multi-objective optimization based neural network for coke furnace system," *IEEE Trans. Ind. Electron.*, vol. 64, no. 4, pp. 3147–3155, Apr. 2017.

[20] W. Dai, T. Chai, and S. X. Yang, "Data-driven optimization control for safety operation of hematite grinding process," *IEEE Trans. Ind. Electron.*, vol. 62, no. 5, pp. 2930–2941, May. 2015.

[21] H. Wang, Y. Jin, and J. O. Jansen, "Data-driven surrogate-assisted multiobjective evolutionary optimization of a trauma system," *IEEE Trans. Evol. Comput.*, vol. 20, no. 6, pp. 939–952, Dec. 2016.

[22] K. Giannakoglou, "Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence," *Int. Rev. J. Progress in Aerosp. Sci.*, vol. 38, no. 1, pp. 43–76, 2002.

[23] Y. Jin and B. Sendhoff, "A systems approach to evolutionary multiobjective structural optimization and beyond," *IEEE Comput. Intell. Mag.*, vol. 4, no. 3, pp. 62–76, Aug. 2009.

[24] J. Yang, J. Dong, and L. Hu, "A data-driven optimization-based approach for siting and sizing of electric taxi charging stations," *Transp. Res. C, Emerg. Technol.*, vol. 77, pp. 462–477, Apr. 2017.

[25] D. Bertsimas, V. Gupta, and N. Kallus, (Dec. 2013). "Data-driven robust optimization," [Online]. Available: https://arxiv.org/abs/1401.0212

[26] M. Mevissen, E. Ragnoli, and J. Y. Yu, "Data-driven distributionally robust polynomial optimization," in *Proc. Adv. Neur. Inf. Process. Syst*, Lake Tahoe, NV, USA: , 2013, pp. 37–45.

[27] L. Wu and A. Markham, "Evolutionary machine learning for RTS game Starcraft," in *AAAI*, 2017, pp. 5007–5008.

[28] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.

[29] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[30] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Sci.*, vol. 220, no. 4598, pp. 671–680, 1983.

[31] Z. Wu, S. He, and X. Yu, "Genetic annealing evolutionary algorithm (in Chinese)," *Journal of Shanghai Jiaotong University.*, vol. 31, no. 12, pp.69–71, 1997.

[32] D. Adler, "Genetic algorithms and simulated annealing: A marriage proposal," in *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, CA, 1993, pp.1104–1109 vol.2.

[33] P.-C. Yip, "The role of regional guidance in optimization: The guided evolutionary simulated annealing approach," Ph.D. dissertation, Dept. Elect. Eng. and Appl. Phys., Case Western Reserve Univ., 1993.

[34] G. Synnaeve and P. Bessiere, "A dataset for Starcraft AI & an example of armies clustering," in *Proc. AIIDE Workshop AI Adversarial Real-Time Games*, 2012, vol. 2012, pp. 25–30.

[35] M. S. Fjell and S. V. Møllersen, "Opponent modeling and strategic reasoning in the real-time strategy game starcraft," M.S. thesis, Dept. Comput. Sci. and Info. Sci., Norwegian Univ. of Science and Technology, Trondheim, Norway, 2012.