Multi-Objective Assessment of Pre-Optimized Build Orders Exemplified for StarCraft 2

Matthias Kuchem
Computational Intelligence Group
TU Dortmund
matthias.kuchem@tu-dortmund.de

Mike Preuss
Computational Intelligence Group
TU Dortmund
mike.preuss@tu-dortmund.de

Günter Rudolph
Computational Intelligence Group
TU Dortmund
guenter.rudolph@tu-dortmund.de

Abstract—**Modern *realtime strategy* (RTS) games as StarCraft 2 educe so-called metagames in which the players compete for the best strategies. The metagames of complex RTS games thrive in the absence of apparent dominant strategies, and developers will intervene to adjust the game when such strategies arise in public. However, there are still strategies considered as *strong* and ones thought of as *weak*. For the Zerg faction in StarCraft 2, we show how strong strategies can be identified by taking combat strength and economic power into account. The multi-objective perspective enables us to clearly rule out the unfavourable ones of the single optimal build orders and thus selects interesting openings to be tested by real players. By this means, we are e.g. able to explain the success of the recently proposed *7-roach* opening. While we demonstrate our approach for StarCraft 2 only, it is of course applicable to other RTS games, given build-order optimization tools exist.**

I. Introduction

Analysing and designing *realtime strategy* (RTS) games may be one of the ultimate challenges for AI game research as these games are highly complex. At the same time they attract millions of gamers – thus they also have considerable economic impact, although their relative importance has weakened in recent years. One of the interesting by-products of famous strategy games as e.g. StarCraft 2 is the emergence of a lively community that drives forward the *metagame*. Simply put, the metagame is what happens outside of the game, but in case of RTS games it is mostly about strategies and counter-strategies. Successful strategies *emerge* and are later discarded again when good counters are found. We state that this strategy-finding is a complex optimization problem which can partly be automated. The most simple, very restricted part of it considers only openings, namely the utilised build orders. The overall task of this work is to demonstrate how build order optimization can be combined with concepts from multi-objective optimization in order to obtain a recommendation system for good openings, and also explain why the recently discovered *7-roach* rush[1] is good. Another, related use of build order optimization by means of coevolution in order to prevent too dominant strategies within the design process of a game has recently been reported for balancing the new game *City Conquest*. [2]

It seems that research on build orders for *realtime strategy* (RTS) games has only recently become an object of academic interest. However, this is undoubtedly an important area at least for the players. It should be also for the game design process, for at least two reasons:

- Long-time player satisfaction with an RTS game partly depends on the possibility to explore and experiment with new strategies (community building).
- AI controlled players (bots) shall be able to apply reasonably good build orders to ensure that defeating them is not too easy.

Existing approaches mostly consider the planning process towards a given target as [1] and its predecessors [2] and [3], or concentrate on matching openings with respect to the opponent strategy [4]. [5] employs *case-based reasoning* (CBR) to extract building sequences from StarCraft replays. Interestingly, [6] couples the strategy planning process with build order optimization.

Our approach acts on a different level, as we are interested in comparing already optimal build orders in a multi-objective fashion. It could be used as a decision aid *on top* of the previously mentioned works, ruling out build orders that may be optimal, but should not be applied because one could get much more in about the same amount of time (e.g. economical power). We do not try to find interesting build orders in replays of human games but conversely provide a tool for humans to detect new ones. Although we focus explicitly on the Zerg faction in StarCraft 2, the approach is (given suitable tools for optimizing the build order) transferable to other factions and games whenever there exists an *economic* component that builds the basis of military power.

At a first glance, one may ask why the economic perspective is important at all when build orders for rush strategies[3] are optimized. If only an *all-in* use of

[1]originally posted by the author of EvoChamber on http://www.teamliquid.net/forum/viewmessage.php?topic_id=160231

[2]http://aigamedev.com/open/interview/evolution-in-cityconquest/

[3]rushes use units early available in the game to start a quick attack in order to destroy the enemy base before it can be fortified

978-1-4673-5311-3/13/$31.00 ©2013 IEEE

these strategies is considered (build combat units and try to annihilate the enemy at a predefined point in time without a backup plan), economy can surely be neglected. However, there are good reasons to take it into account:

- Players who opt for rush strategies need to deliver a certain amount of damage to the opponents even if these cannot be defeated at once. In case both survive (which is not unusual), the ratio of own economic power to the remaining economic strength of the opponent is crucial for winning the game.
- In a game with more than 2 players, rushing is very risky and if it fails, one is thrown back to further development, for which the economical basis is decisive.

In a nutshell, our approach is to detect the quickest build orders to obtain a certain number of attack units over a whole variety of unit types and numbers and some popular 2-type unit combinations, record their economic power (number of worker units) and then analyse the results in a multi-objective fashion by ruling out *dominated* strategies.

However, domination shall not be understood as traditionally used in multi-objective optimization. Originally, we have 3 objectives here: military unit number (combat strength) and worker unit number (both to be maximized), time (to be minimized). Each solution we consider is itself the result of an optimization process, namely of finding the quickest build order for a given military unit number. By using this optimality, we can ensure that combat strength increases monotonically over time, which in turn can be used for altering the domination concept and effectively reduce the number of considered objectives to 2. The modification of the domination concept is very simple: solutions only dominate others if they provide more combat strength (equals higher build time) AND higher economic power. One could argue that a higher number of attack units shall always come with a larger amount of workforce, but StarCraft 2 is obviously too complex for such a simple relation, and our results show that this is indeed not the case.

We now apply the basic idea of evolutionary multi-objective optimization to the outcome of our systematic build order study: keep only the non-dominated (Pareto optimal) population, or in other words, the ideal compromises. The basic notions of (evolutionary) multi-objective optimization can be grasped from [7], chapter 9.

This work does not focus on new fancy AI algorithms but puts together existing methods in a new fashion in order to support and understand RTS metagames. Our contribution is exactly the architecture that brings these game-specific parts together and couples them by means of multi-objective optimization, which is of course easily done also for other games if an appropriate build order optimization tool exists.

Here, we use as build order optimization tool a freely available method (EvolutionChamber by Ronald Ray (aka Lomilar)[4] which uses a *genetic algorithm* (GA) based approach) in a different way than it was originally conceived. Here, we concentrate on build order properties of the Zerg faction and present some interesting insights on this matter.

However, our analysis should not be interpreted as the last word on Zerg build orders as due to the nature of the employed method and the vastness of the search space we cannot guarantee optimality of the results. We would not expect that our results on rush build orders can be improved by a large margin, but we completely neglect an important factor: We abstain from tackling relevant non-rush (tech and late-game focused) build orders as this would be a much more difficult endeavour.

Fig. 1: Zerg base attacked by Protoss units

We now start by providing some background informations on StarCraft 2, and then introduce the employed EvoChamber tool in section III. The basic ideas of our multi-objective approach are containted in section IV, followed by the display and discussion of the obtained results (section V), a technical discussion that evaluates the EvolutionChamber tool, and the conclusions.

II. StarCraft 2 primer

StarCraft 2 (see figure 1) was published in 2010 as long-awaited successor of the extremely popular real-time strategy (RTS) game StarCraft of 1998. Besides improved graphics, gameplay and opponent AI, the basic game mechanisms have not been changed: There are three very different factions, each with their own build processes, tech trees, units, and their specific advantages

[4]http://code.google.com/p/evolutionchamber/

and disadvantages. These factions are: Terran (human-like), Zerg (biomorph), and Protoss (psionic). Game mechanics differ quite a lot for the factions, e.g. most Terran buildings can fly (very slowly), and Zerg buildings are not produced, Zerg drones (workers) morph into the desired buildings.

The balancing of the factions is remarkably stable, so that even on the grandmaster level (top 200 players of every region, current regions are: Americas, Europe, Korea, Taiwan, Southeast Asia), there are no clear preferences for one faction over the others over a longer period of time. However, the metagame from time to time leads to the discovery of a strong strategy that favors one faction if played well and not countered early. Over time, counter-strategies come up which remove again the temporary advantage of that faction. In this work, we concentrate on the Zerg only as the employed build order optimization tool (EvolutionChamber) cannot deal with the other races. But in principle, the same approach could be used for Terran and Protoss.

The Zerg race starts the game with a hatchery, 6 drones and an overlord. The hatchery is the main building which produces larva that can morph into most units. Drones are the workers who gather ressources from mineral patches and extractors on vespene gas geysers. Each overlord provides 8 supply, limiting the maximum number of units. Besides the amount of gathered resources, supply and larva are the limiting factors for unit production. The larva production on hatcheries can be increased with queens.

There are several Zerg units available, all with different tech-trees. Here, we introduce the ones covered in the presented build orders.

Zergling The first army unit in every Zerg tech-tree. Two zerglings spawn from each larva. They are weak in attack and defense, but cheap and early available. With speed upgrade, they are the fastest unit in StarCraft 2.

Roach Relatively quickly available and cheap, with simple tech-tree. Possesses low damage rate, but good defense.

Mutalisk Fast flying units, mostly used to harass enemy workers and very useful against all units without anti-air attack. Mutalisks have an expansive tech-tree and take time to produce.

It is also possible to upgrade weapons and armor, which is considered on the roach unit in the second build order. More detailed information on all units, resource costs and build times are provided by Liquipedia [8].

A *build order* is a list of ingame actions. As an example, we list the shortened version of the generated 7 roach rush build order below (build order II.1). The reason for looking after a fixed build order is to guide players through the opening, which is otherwise pretty complex due to the number of possible actions and the difficulty to detect a good path even for a clearly defined goal. When we speak of usually considered build orders for Zerg players, we refer to professional gamers, competing in the big tournaments as GSL, MLG, IPL, IEM and Blizzard Invitational. How do these players generate their opening build order? Their basic idea is to produce as many drones as possible, while teching up for the unit composition of their choice. But all testing and optimization is done by experimenting and playing; although it is possible to calculate the efficiency of specific build-orders. This is due to the lack of feature-rich analyzation tools and the necessary computation time.

0:51	10/10	extractor trick
1:04	11/10	overlord
1:31	11/18	spawning pool
2:04	14/18	extractor, send on 2:39; 2:56; 3:07
2:39	16/18	roach warren
2:56	15/18	queen
3:07	17/18	overlord
3:25	18/18	overlord
3:34	18/26	extractor trick
3:39	18/26	7 roaches

Build order II.1: generated build order for 7 roaches, shortened by not listing drone builds, these are encoded in the second column

As of 2013, the focus of Zerg players is to strengthen the economy as much as possible, while maintaining map control and pressure on the enemy. Therefore, most of their build orders are late- or mid-game focused while our build orders focus on early game dominance. However, it is part of the current metagame to start an early game focussed build order when your enemy expects a drone focused build to punish this assumption and the corresponding build decisions. This follows the reasoning to confuse the enemy by pressuring him and hurt his economy. Sometimes it is even possible to win the game right away.

III. Optimization via EvolutionChamber

Our basic tool for detecting (near-)optimal build orders is the EvolutionChamber program which only deals with the Zerg faction of StarCraft 2. SCfusion[5], another similar and more efficient tool unfortunately lacks a command line interface and can thus not be automated. Every single build order optimization (of which we have done thousands) must be initiated manually via its GUI.

EvolutionChamber is based on the JGAP [9] library that provides generic GA and *genetic programming* (GP) components. EvolutionChamber is started by providing waypoints that consist of a maximal time and a targetted set of units/buildings/ technologies. It then computes the strategy that produces this set in the shortest possible

[5]available at http://code.google.com/p/scbuildorder/

time, with the number of drones (Zerg workers) as a second objective. We use it with one waypoint only, but several would be possible.

Internally, EvolutionChamber employs a fixed length (based on the input goals) encoding for every individual (chromosome) that lists the different build commands and other actions in the order that is then simulated, including units, buildings and technologies. The list of possible actions also include special actions as e.g. the extractor trick[6], mining and waiting. During the simulation phase, these encoded actions are carried out, starting from the usual set of a hatchery, six drones and one overlord (which is given and not evolved), and evaluated by means of a fitness function which is described in the following.

A. Fitness function

The game state resulting from the simulation is compared to the target defined by the provided waypoint. For each unit/ building/technology in the waypoint list that was actually built its resource value (minerals plus vespine gas) is added up. For every other built unit, the hundredth of this value is added, for additional technology buildings (if one of this type already exists) this value is halved again. The next step is to test if the target has been reached. If this is not the case, only some points are added for the remaining resources (with a factor of ≈ 0.001). For every non-executable encoded action, the result is decreased by one point.

If the target has been achieved, a constant bonus of 500 points is added and the remaining resources are converted into points with a higher factor of ≈ 0.01. More importantly, the time saved with respect to the given target time is added using the following formula: $\left({}^{\text{target time}}/_{\text{needed time}}\right)^2$.

Unfortunately, we do not know the design criteria of the fitness function. However, we find that the most important factor is the time needed to reach the target, and that the number of units/buildings/technologies during this time is maximized. As drones get a higher weight than all other units, maximization of the economic power can be seen as implicitly given as a secondary objective.

B. Genetic operators

EvolutionChamber uses a number of operators to generate new individuas (chromosomes), some of which are very problem-specific. Note that we did neither design nor modify these operators, although there may be room for improvement here. They are executed in the following order:

cleaning operator Looks for the current best individual and uses it as basis for creating n new ones, where n is the current length of the individual. Within each new individual, the ith gene (i runs from 1 to n) is left out. The idea behind is that superfluous actions can be deleted.

overlord operator This operator works almost like the cleaning operator, however, instead of deleting the ith gene it is replaced by the action *build overlord* to ensure that the unit limit is high enough for the other encoded actions to actually be executed.

insertion operator Copies randomly chosen individuals and adds a random action at a random position.

removal operator Copies randomly chosen individuals and randomly deletes one action from the copy.

rotation operator Copies a randomly chosen indivdual and reverts the order of two consecutive genes.

exchange operator Two genes of an individual are chosen and a new individual is created that contains a copy with the two genes exchanged.

[6]By starting to morph a drone into an extractor, the unit counter can be decreased so that a new drone can be built. Cancelling the morph then leads to a situation where the number of units is actually one higher than the maximum number, which depends on the number of existing overlords.

C. Setup and stopping criterion

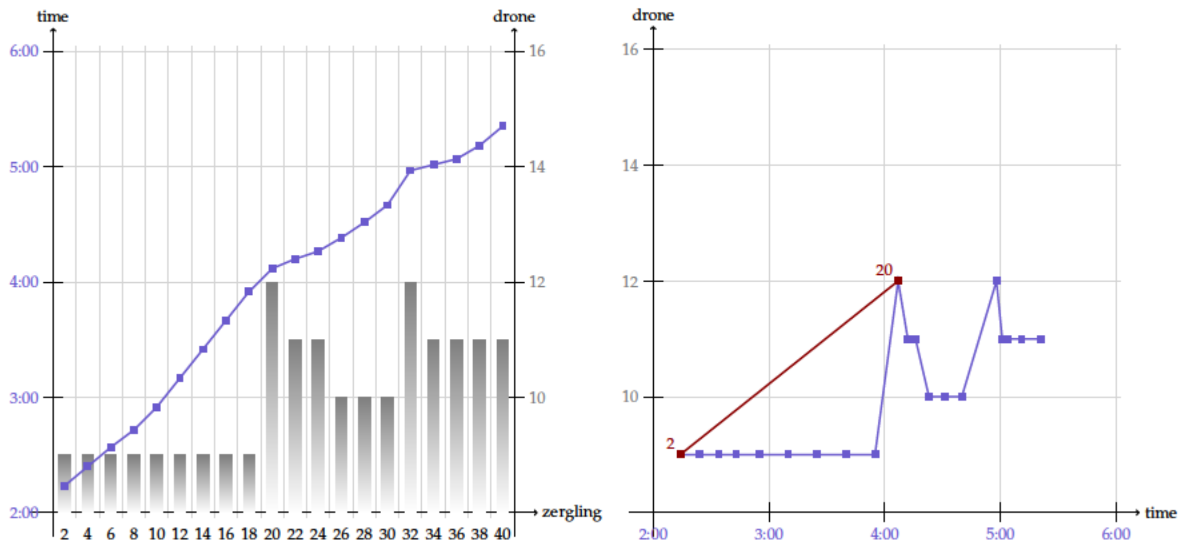
For every single target (sought unit combination, see below), we executed 10 runs with 8 concurrent optimization processes. A run was stopped when no further improvement was found for more than 400 generations within each process. In consequence, every target was pursued 80 times. The population size for each run was set to 400 after some manual testing.

IV. Multi-objective Analysis

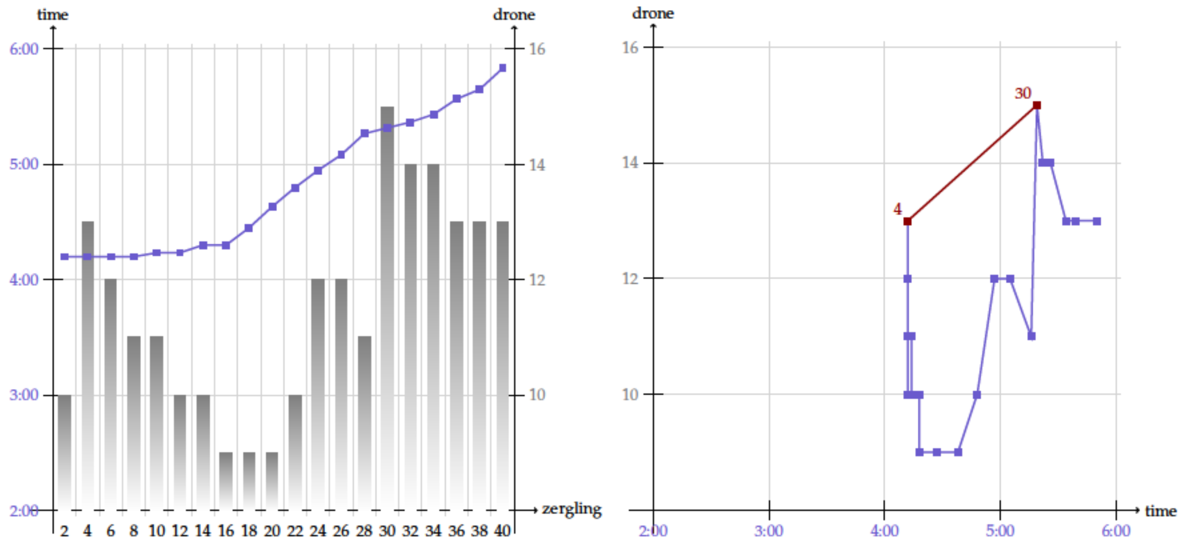
As already stated in the introduction, we are interested in the dominant strategies in the sense that these form a Pareto front between the objectives of more attack and more economic power. However, we demand that a dominating solution is better in both objectives. This of course means that quicker build orders shall never dominate longer ones (as both are (near-)optimal with regards to the number of combat units produced). In the standard multi-objective formulation of the dominance principle, the latest solution would dominate all others. However, we have to adapt this principle here to take into account that the solutions are not comparable anyway: they are solutions to different optimization problems (build x units vs. build y units where $x < y$).

Furthermore, we are especially interested in quick solutions as rush strategies have the highest probability of success if the attack starts early. A Pareto front between time/combat strength and economic power leads to strong attacks at different timings but always backed by strong economic power. All optimal build orders with relative weak economy are disregarded.

One could also try to take different unit types into account for comparison, but it would be very hard to compare e.g. a zergling rush to a roaches rush because success highly depends on the opponent strategy which is not considered here (and which is also usually not



(a) zerglings without upgrades, no scouting



(b) zerglings with speed upgrade (metabolic boost), no scouting

Fig. 2: (near-)optimal build orders for zergling rush strategies, left figure: attack times (blue dots) and number of drones (gray boxes) over number of units, right figure: same data but another view to visualize the Pareto optimal points (red, numbers mean attack unit count) according to timing (correlating to attack strength) and economic power.

considered by the players unless they have some idea what kind of strategy the opponent will play).

Thus, if we stick to one unit type or a 2-type combination with defined ratio, attack power will grow monotonically with time and we can focus on timing and economic power as the two objectives. To obtain the necessary data for multi-objective reasoning, we run the EvolutionChamber separately as reported in section III-C with the desired number of attack units as target. This number goes from 2 to 40 for zerglings (as always 2 are produced), from 1 to 20 for roaches and from 1 to 10 for mutalisks. The limits have been chosen according to the approximate minimal time for the highest numbers, which is around 8 minutes. It may be disputable if a strategy that strives for an attack after 8 minutes is still a rush, usually these are carried out between 2:30 and 7 minutes.

While the concrete results will be discussed in the next section, we introduce the graphical presentation here, employing figure 2 as an example. The left figure shows the quickest build order obtained for each number of zerglings, where the blue rectangles mean the build time (according to the left axis), and the gray bars the number of drones at the end of the build (according to the right axis). As expected, build times grow monotonously with the number of zerglings. However, the number of drones (economic power) does not show such a clear correlation to the build time. In order to detect the Pareto front between timing and economic power, the right figure depicts only the number of drones over time, and the red rectangles give the non-dominated points, with the according number of attack units plotted as numbers next to the points. The red lines can therefore be interpreted as the Pareto front. We presume that only the non-dominated strategies (with the number of drones growing with the number of attack units) can be recommended as openings as the ones between the points give away a possible economic development.

V. Concrete results

In the following, we present and interpret the results for the unit types zerglings, roaches, and mutalisks.These are probably the most important rush strategies for the Zerg faction. Others could be computed in the same fashion, but we need to restrict the types due to space constraints. However, the exact choice regarding scouting or no scouting and upgrade technology selection may be somewhat subjective.

A. Zerglings

As zerglings are the first attack units that become available and themselves often used for scouting, all investigated strategies abstain from doing extra (drone) scouting beforehand.

a) Zerglings without metabolic boost (speed upgrade): require almost no tech structure. Therefore, rushing with zerglings is the earliest push one can make. In the StarCraft community, the *6 pool* is quite famous. This means that the spawning pool for zergling production is built as early as possible (without developing any more drones) and zergling production is started as soon as it is finished. With respect to our results in fig. 2a, this is a good strategy until more than 18 zerglings are produced initialy. It is important to send the first produced zerglings to the oponents base to prevent a Protoss or Terran oponent from walling their base off with buildings. Therefore, even the second Pareto optimal point with 20 zerglings is not playable vs. Terran or Protoss on most maps (standard 2 or 3 building wall off) as the first attack units spawn too late. On the other hand, this build order is good against any Zerg opponent: The first pair of zerglings spawn shortly after a 6 pooling opponent strategy can reach the own base (2:14 + travel time vs. 2:51). Which means that this strategy is superior to the 6 pooling strategy, being safe against all early pushes while producing more drones and attacking units.

b) Zerglings with metabolic boost (speed upgrade): are far stronger than zerglings without the upgrades: Since they are melee units, they need to surround the enemy unit to deal maximal damage in masses. They are also the fastest units in game on creep, so they can prevent every other unit from escaping. Our results (fig. 2b) on this composition are rather disappointing, since the first Pareto optimal build-order builds 4 zerglings, delayed to 4:12 for speed upgrade and the next Pareto optimal point is on 5:19 with 30 zerglings, which is far too late considering counter-strategies, even for Zerg with roaches (Terran and Protoss will be walled off, first zergling in this build order spawns on 2:47 which is too late to hurt an opponents Zerg economy in order to prevent him from building roaches countering the zerglings). Therefore this composition is a good follow up strategy after a first rush without speed upgrade (since all upgrades apply to the already existing army), but not a good opening.

B. Roaches

Roaches are considered to be mainly an early to mid-game unit, for its simple tech-tree and low resource costs. The basic goal of a rush is to reach the opponent's base before it is heavily guarded. Therefore, it is possible to scout with the first roaches, which arive at a good scouting timing on the enemies base (although not scouting with drones always bears the risk of not detecting a 6 pool enemy strategy with its weak but very quick rush).

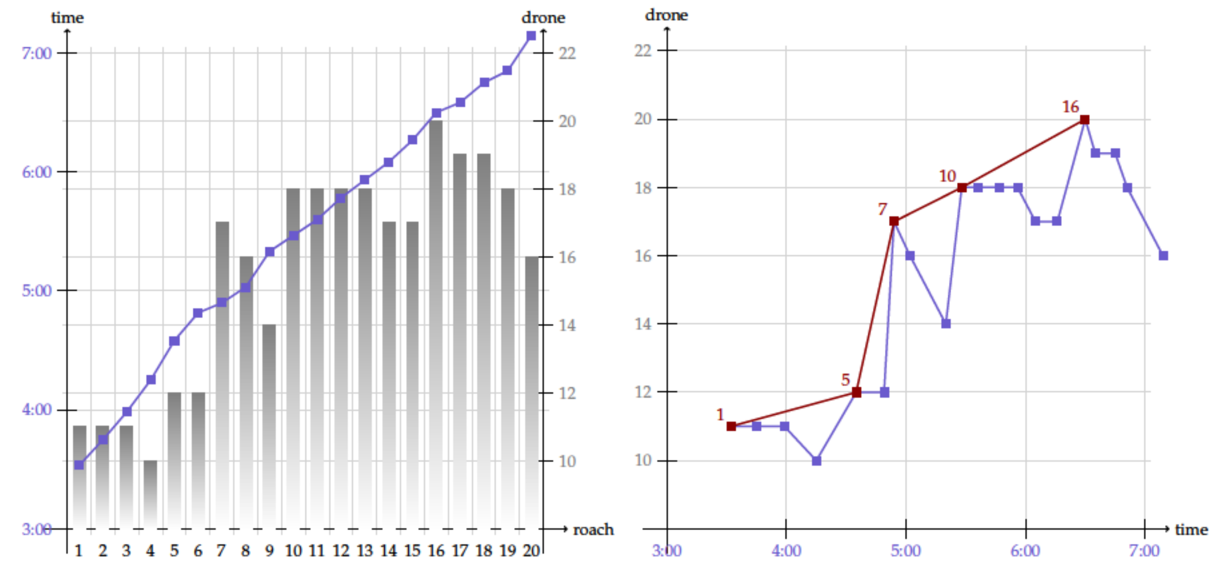
Interestingly, roach rushes are very popular within the StarCraft community. For example, in 2010 a user suggested a 5 roach rush on the TeamLiquid forums[7], describing it as a solid build (not all-in), because of its good economic power. In fig. 3a, we indeed find this build as one of the Pareto optimal points. A few months later, a 7 roach rush strategy became famous as a very strong push with very good economy. This build was generated with the EvolutionChamber by its author, searching for an even stronger build than the 5 roach rush[8]. The results depicted in fig. 3a also support this strategy as it is Pareto optimal. Furthermore, it is not very likely that a good 8 or 9 roaches rush will be discovered. The other two Pareto optimal points (10 and 16 roaches) may be strong pushes, but they reach at the opponents base much too late, when it will most likely be already well defended (a 7 roach rush normaly arrives right at the time when defense is in production).

C. Mutalisks

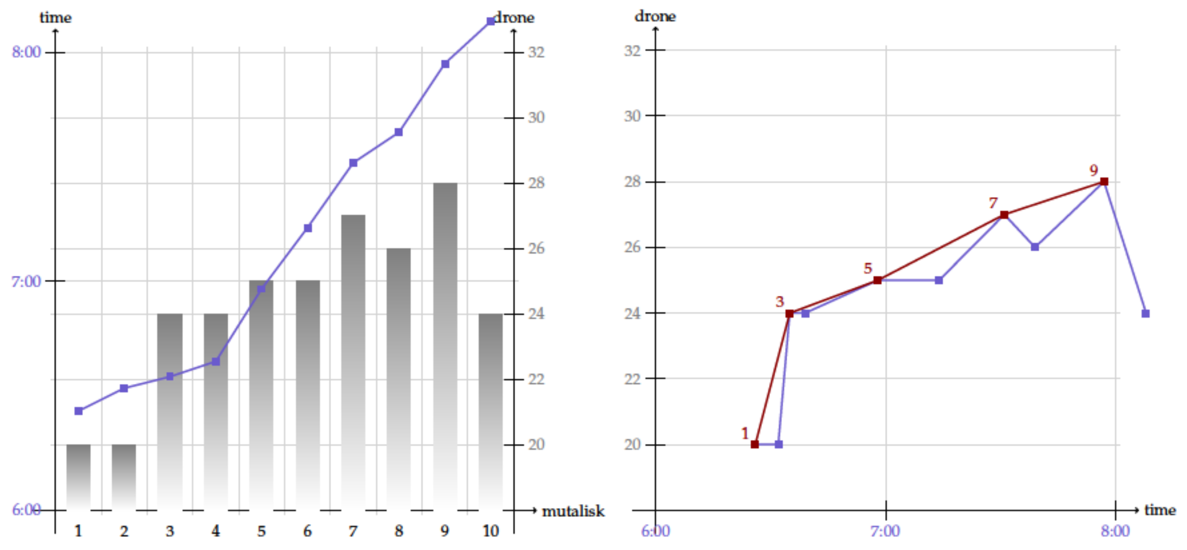
As mutalisks appear relatively late in the game (as do hydralisks), scouting appears to be mandatory. Our results in fig. 3b provide several options with 5 Pareto optimal builds out of 10. The first one can be dismissed as a single unit is not strong enough to do any reasonable damage within a given time. Besides, going for

[7]http://www.teamliquid.net/forum/viewmessage.php?topic_id=145740

[8]http://www.teamliquid.net/forum/viewmessage.php?topic_id=160231



(a) roaches, no upgrades, no scouting



(b) mutalisks with scouting

Fig. 3: roach and mutalisk rush strategies, left: attack times (blue) and workers (gray) over number of units, right figure: Pareto view of same data (non-dominated points in red), according to timing (correlating to attack strength) and economic power.

3 mutalisks takes only 9 seconds more. The next step of 23 seconds to 5 mutas is bigger and clearly leaves an option to try both. Due to the nature of the mutas attack type (hitting multiple targets), 2 more mutas will increase the damage on opponents economy heavily and the 5 muta rush appears to be favourable. In this respect, it is important to consider the point in time when the opponent starts to build anti-air defense. For the Terran faction this will usually happen between minutes 8 and 9 (It would be possible to build anti-air earlier, but this is default timing of incoming mutas). The 5 mutas spawn at around 7 minutes, leaving at least a full minute to fly to the opponents base. The other two strategies with 7 and 9 mutas probably come too late and their attacks will meet a well defended base.

VI. Technical Discussion

While our approach to use EvoChamber as a tool within much larger, systematic experiments (not just for detecting build orders for one concrete target) obviously produced some interesting insights, we cannot know how far each single best build order is from the optimum. We presume that the results are already quite good as it has not been possible to improve them by massive use of computational resources (many restarts, parallel runs). However, it is clear that good build order optimization is very dependent on the use of specific techniques (e.g. the extractor trick), and if these are left out, it will not be possible any more to reach the best solutions.

This may also be the reason why EvoChamber only utilizes the Zerg faction; transferring it to other factions or even games not only means to load another table of unit properties but also to respect many game-specific subtleties which prevents us from obtaining a general build order optimization method as well as from easily transferring the results to other games.

From the application point of view, EvoChamber is surely not optimal in many ways, e.g. we obtain many superfluous extractor tricks. The utilized search operators seem to be sufficient to enable reaching good solutions, but our impression is that they are not very efficient so that a lot of computational power is wasted. The fixed length representation is also problematic, because the limit could be chosen too low to enable some very complex but good build orders, and for the more simple ones it means unnecessary effort. The employed fitness function (encoded in EvoChamber) may also be revised, as there are several constants with an ad-hoc character. Besides only evaluating the results of a lot of single-criterion optimization runs, one could also think of a pure multi-objective approach that should be able to obtain the non-dominated solutions much quicker, preferrably in a single run. The results of this paper could then serve as a basis for evaluating this approach.

Despite the many points of critisism we have named, we are grateful that the EvoChamber system exists and works pretty well.

VII. Conclusions and Future Work

We have applied an existing StarCraft 2 build order optimization tool in a completely new way, motivated by the inherently multi-objective perspective of the latest findings in the StarCraft 2 metagame (e.g. 7 roach rush). To our knowledge, there are currently no scientific works concerned with analysing RTS metagames to the point that new, most likely very good openings (or their build orders) can be detected. Our approach largely automatises this (still, many ingredients have to be assumed or provided as expert knowledge and cannot be automatically detected).

By systematically investigating the best build orders for meaningful numbers of attack units and also reviewing their economic component, we provide an insightful and new comparison of opening strategies that may be interesting for players as well as for AI designers, and a (meta-)method that may be transferred to other RTS games. It shall also be interesting to develop this approach further to provide the Pareto optimal solutions in one run, but the results would be more or less the same, this would only reduce the computational cost (probably dramatically).

Taking into account the concrete results for the different unit types, it seems that there is a basic difference between the simpler ones for the early units (zerglings, roaches) and the later units (mutalisk, hydralisk). The build orders for the latter apear to be more stable, with fewer dramatic shifts. For the zerglings, we can recognize such a strategy shift from 18 to 20, and the reason is that in order to reach 20, a queen has to be produced to increase the larvae production rate. From this example we can conclude that StarCraft 2 build orders are fairly complex and not easily scalable even for relatively simple targets, which shall be seen as an advantage of the underlying game.

Besides a pure multi-objective approach, many extensions of the current work can be imagined, of which the most tempting may be comparing different unit types. While this may be very difficult, one could do a small step into that direction e.g. by comparing simple units with their upgraded versions to see when and under which conditions upgrades pay off.

References

[1] B. G. Weber and M. Mateas, "Case-based reasoning for build order in real-time strategy games," in *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2009, October 14-16, 2009, Stanford, California, USA*, C. Darken and G. M. Youngblood, Eds. The AAAI Press, 2009.

[2] A. Kovarsky and M. Buro, "A first look at build-order optimization in real-time strategy games," in *Proceedings of the GameOn Conference, Braunschweig Germany*, 2006, pp. 11–22.

[3] H. Chan, A. Fern, S. Ray, N. Wilson, and C. Ventura, "Online planning for resource production in real-time strategy games," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2007, pp. 65–72.

[4] G. Synnaeve and P. Bessière, "A bayesian model for opening prediction in rts games with application to starcraft," in *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31 - September 3, 2011*, S.-B. Cho, S. M. Lucas, and P. Hingston, Eds. IEEE, 2011, pp. 281–288.

[5] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *IJCNN*, 2008, pp. 3106–3111.

[6] D. Churchill and M. Buro, "Build order optimization in starcraft," *Proceedings of AIIDE*, pp. 14–19, 2011.

[7] A. Eiben and J. Smith, *Introduction to evolutionary computing*, 2nd ed. Springer, 2007.

[8] Team Liquid, "Liquipedia, the StarCraft II encyclopedia," 2012. [Online]. Available: http://wiki.teamliquid.net/starcraft2/Zerg_Strategy

[9] K. Meffert, "JGAP java genetic algorithms package," 2012. [Online]. Available: http://jgap.sourceforge.net