# Wasp-Like Scheduling for Unit Training in Real-Time Strategy Games

**Marco Santos** and **Carlos Martinho**
INESC-ID and Instituto Superior Técnico, Technical University of Lisbon
IST Taguspark, Av. Prof. Dr Aníbal Cavaco Silva, 2744-016 Porto Salvo, Portugal.
{marco.d.santos, carlos.martinho}@ist.utl.pt

**Abstract**

Gameplay in real-time strategy games seems somehow to be confined to a de facto standard where economical micro-management is equally important as combat strategy, if not more important. To enable stronger combat-oriented gameplay without sacrificing other key aspects of the genre, we propose an automated system for scheduling unit training, which we believe may allow the exploration of new paradigms of play. To be accepted by the player, such a system must, among other things, be efficient and reliable, which is a non-trivial task considering the highly dynamic nature of the environment in this genre of games. To overcome such a challenge, we propose a system inspired in the swarm intelligence demonstrated by social insects, namely wasps, and describe its limitations and benefits, based on the evaluation of an implementation of the approach as a mod(ification) of the game Warcraft III The Frozen Throne.

## Introduction

When analyzing the evolution of the real-time strategy (RTS) genre over the past years, gameplay seems somehow to be confined to a de facto standard: collect resources, construct a base and an army, overcome the enemy and repeat. Some games, such as Company of Heroes (Relic Entertainment, 2006) or Ground Control (Massive Entertainment, 2000), break away from this “formula” and stand out due to their combat-oriented gameplay, which is achieved at the expense of other key aspects of the genre, such as economic management. We believe that by offering mechanisms to automate micro-management of certain aspects of the game (for instance, individual unit training), we will promote the exploration of novel paradigms of gameplay for such genre of games, by allowing the player to choose which aspects to micro-manage while maintaining a macro level control on the other aspects of the game. In this paper, in particular, we explore an approach where the player only specifies the position where an unit is requested. This relieves her from micro-management routines, such as having to specify where each unit is produced or trained and in what order. In this case, the macro level control is maintained by requiring the player to specify where each production or training facility is built in the game environment.

To earn the trust of the player and not hinder game experience, the system supporting the automation of micro-management tasks has to be efficient, robust and reliable within the ever changing environment that characterizes this genre of games. To deal with the highly dynamic nature of the environment, we explore a solution based on the behavior of social insects capable of producing complex emergent behavior as a colony, responding and adapting themselves to external perturbations in a decentralized manner. Such mechanisms have already been shown capable of handling intricate engineering problems (Cicirello and Smith, 2001).

This document is organized as follows. First, we discuss the adequacy of insect behavior and particularly wasp behavior to tackle the problem of real-time scheduling in RTS games. Next, we describe WAIST, the wasp-inspired algorithm we implemented in the game Warcraft III The Frozen Throne (Blizzard Entertainment 2003), as well as the novel gameplay concepts we introduced in our evaluation scenario. Then, we present the evaluation methodology and report the results achieved by WAIST in different variants of our scenario. Finally, we draw some conclusions on the benefits and limitations of the approach and present some possible future directions for our work.

## Related Work

### Swarm Intelligence

Swarm intelligence is a term referring to the collective emergent behavior resulting from decentralized and self-organized of natural and artificial systems (Beni and Wang, 1989). Its roots are the studies of self-organized social insects, such as ants, wasps or termites. Although such insects have strict sensory and cognitive limitations, the colonies manage to perform complex tasks such as foraging (Traniello, 1989), brood clustering (Sendova-Franks et al., 2004), nest maintenance and nest construction (Perna et al., 2008). The mechanisms underlying their complex emergent behavior is the subject of great interest and study, resulting in a wealth of nature-inspired models (Engelbrecht, 2006). Of relevance to our work, the problems dealt within a colony are analogous to the scheduling and logistic engineering problems addressed by Man. We are particularly interested in the wasp task allocation behavior.

## Task Allocation of Wasps

From their studies of the *polistes dominulus* wasps, Theraulaz et al. (1991) created a model of dynamic task allocation that emulates the self-organized behavior of wasps. The system has the following main features:

- Tasks have the capacity of emitting *stimuli* that affect the individuals' task selection decisions;
- Individuals possess *response thresholds* that represent their predisposition to perform certain tasks;
- Each individual has a *force* that is taken into account during dominance contests to determine the winner;
- When an individual performs a certain task, its response threshold changes to increase the predisposition to repeat the task, creating *specialists* in the society.

These four features drive the model both toward performance and flexibility. The system self-organizes itself towards optimal performance due the individuals' capacity of specialization. This specialization, however, is not rigid, allowing them to dynamically adapt their work force according to the constantly changing environment (e.g. loss of individuals). Such characteristics are of importance when considering the needs of real-time scheduling for RTS games.

## Routing-Wasp

Based on the properties of the natural model created by Theraulaz et al. (1991), Cicirello and Smith (2001) proposed an algorithm for dynamic task allocation that later was adapted to Morley's factory problem from General Motors (Morley, 1996), denominated Routing-Wasp or R-Wasp.

In this algorithm, each different task is capable of emitting stimuli which *strength* if proportional to the time the task has remained unassigned: the longer a task remains unassigned, the stronger the emitted signal. Each agent has a *response threshold* associated with the tasks it is capable of performing, representing its propensity to *bid* for the task: the lower the threshold, the higher the propensity to bid. For each unassigned task, agents stochastically decide to bid or not according to the strength of the emitted stimulus and the response threshold associated to the task.

After all the agents have decided to bid or not, and if there is more than one candidate, a *dominance contest* occurs. The dominance contest consists in a tournament where duels are made until one last standing agent wins. The winner of a duel is stochastically settled taking into consideration the agent forces, which vary according to their suitability to perform the task. The task is then assigned to the dominance contest winner.

Finally, response thresholds are updated. The more an agent performs tasks of a certain type, the more the threshold decreases and the more likely it will accept other tasks of the same type. Reversely, the less an agent performs a certain type of tasks to the detriment of others, the less likely it will accept tasks of that type. Additionally, if an agent is not performing any task, all thresholds are gradually lowered, increasing the propensity to accept any task.

The R-Wasp strength lies in two aspects. First, response thresholds allow the creation of task specialists and gives them the ability to bid for the tasks most suited for them, while allowing them to do other tasks if needed. Second, force permits a fair distribution of the workload between the agents by taking into consideration their characteristics for the task when determining the winner. These two aspects, together, allow the system to self-regulate and dynamically respond to unexpected events like variations in demand.

Such characteristics are relevant for real-time scheduling in the RTS game genre. As such, in this work, we extended and adapted the R-Wasp algorithm to the RTS game domain and evaluated its adequacy.

## Scenario

Before detailing our adaptation of R-Wasp to RTS games, we will first describe the evaluation scenario. We implemented a modification (mod) of the video game Warcraft III The Frozen Throne (Blizzard Entertainment, 2003) providing the player with a new type of interface to request units. In the base game, to build a new unit, the player has first to build a "unit factory" and then issue build orders individually for each unit. Each type of factory is only able to produce a limited number of units and unit types, and different factories produce different unit types. As such, the player has to micro-manage unit production and constantly move between factories, while controlling all the other aspects of the game simultaneously.

Our scenario provides the player with a macro-level management of unit production and explores a new paradigm of gameplay. The player still decides which factory types to build and where to build them but, afterwards, only needs to specify the location where a certain amount of units of certain types are requested, and the underlying system will devise a near-optimal production schedule accordingly. As such, we ensure there are still meaningful choices for the player to make regarding resource management, albeit at a completely different level.

Since scheduling is automated and does not hinder cognitively the player, we provided additional interesting choices, by refining the production model and introducing unit heterogeneity in factory production, i.e. different factories are able to produce the same unit type but have different production times, and an extra setup time is required when production changes from one type of unit to another.

Specifically, in our evaluation scenario (see Fig. 1):

- Only two types of units are considered: $F$(ootman) and $R$(ifleman);
- Changing from the production of one type of unit to another results in extra setup time before resuming production;
- Five different factories are available to the player (see Fig. 1), each one with different balanced properties:
  - Two specialized factories ($F_F$ and $F_R$) are efficient at producing one type of unit ($F$ and $R$ respectively) but unable to produce the other.
  - One factory ($F_H$) is able to produce both unit types but not as efficiently as the specialized factories. Setup time, however, is very high.

- Two other factories ($F_{FR}$ and $F_{RF}$) are also able to produce both unit types and have a lower setup time than the hybrid factory. However, one of the unit type has a high production time.

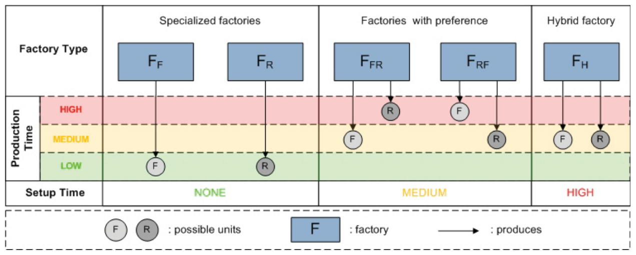

Figure 1: Representation of the scenario factories. As an example, $F_{FR}$ can produce $F$ and $R$ but $R$ will take more time. If after producing a $F$(ootman) the player orders a $R$(ifleman), a medium setup time is required before the production of $R$ begins.

This scenario will by used to test the system's response to a series of unit requests using a preset of factories.

## WAIST Model

We will now describe the algorithm used to implement real-time scheduling in our scenario. The pseudocode for the algorithm is depicted in Fig. 2.

```
var: list of unassigned orders
var: list of factories
var: list of candidates = {}
for-each unassigned order
    for-each factory
        decide to bid or not              (1)
        if( bid )
            candidates <- factory
    dominance contest between candidates (2,3)
    assign order to the winner
    update forces                         (4)
update thresholds                         (5)
update stimuli                            (6)
```

Figure 2: Pseudo-code of the algorithm. The marked steps are detailed in the main text.

Each unit requested by the player is a task that must be performed. Tasks have a type, which is defined by the type of the unit to be produced (in our scenario, Footman or Rifleman), and are capable of emitting stimuli perceptible by all existing factories.

Each factory $f$ capable of producing units of a certain type is conceptually modelled as a *wasp* capable of executing production tasks and has a response threshold $T_{f,t}$ associated with this task $t$. When a factory $f$ receives a stimulus $S_t$ from a task $t$, it stochastically decides to bid or not to bid according to the probability defined by Eq. 1.

$$P(bid \mid T_{f,t}, S_t) = \frac{S_t^2}{S_t^2 + T_{f,t}^2} \tag{1}$$

If no factory bids for the task, the task remains unassigned until the next cycle. If only one factory bids for the task, it is attributed to it. If more than one factory bids for the task, the winner is selected after a tournament of dominance contests.

First, if an odd number or factories bid, the number of participants is made even: $N$ factories are automatically passed to the next phase of the tournament according to Eq. 2, in which $C$ stands for the number of competitors.

$$N = 2^{\lceil log_2(C) \rceil} - C \tag{2}$$

Each dominance contest takes the resistance of the biding "wasp factories" into consideration. The probability of factory $f_1$ winning over factory $f_2$ in a duel is given by Eq. 3, in which $R_{f_1}$ and $R_{f_2}$ represent the resistance of each factory respectively.

$$P(f_1 wins \mid R_{f_1}, R_{f_2}) = \frac{R_{f_2}^2}{R_{f_1}^2 + R_{f_2}^2} \tag{3}$$

The resistance $R_f$ of a factory $f$ is given by Eq. 4, where $T_{prd}$ is the sum of all the production times of the tasks currently in the factory's queue, the task being processed and also the one being disputed; $T_{stp}$ is the sum of all required setups, considering the task being disputed; and $T_{dlv}$ is the estimated time the unit will need to reach its target from the factory (i.e. to be "delivered"). The lower the resistance, the faster the factory will execute the task.

$$R_f = T_{prd} + T_{stp} + T_{dlv} \tag{4}$$

In each cycle, all response thresholds of each factory are updated according to the following function:

$$T_{f,t} = \begin{cases} T_{f,t} - \delta_1 & \text{if the last task in factory } f \text{ queue is of type } t \\ T_{f,t} + \delta_2 & \text{if the last task in factory } f \text{ queue is not of type } t \\ T_{f,t} - \delta_3 & \text{if factory } f \text{ queue is idle} \\ T_{f,t} - \delta_4 & \text{if factory } f \text{ queue is idle and there is an unassigned task of type } t \end{cases} \tag{5}$$

$\delta_1$ acts as a "learning" coefficient, since it diminishes the threshold and encourages the factory to take tasks of the same type, as opposed to $\delta_2$, which acts as an "unlearning" coefficient. These two rules are important to increase the sequencing of orders in the factories' queues and, as a result, reduce the number of setups. When factories are idle, their response thresholds are slowly decreased by $\delta_3$ if there are no unassigned tasks, and more rapidly decreased by $\delta_4 > \delta_3$ to ensure factories do not remain inactive if there are still tasks to perform. Thresholds are also clamped to a maximum range.

Task stimuli are also updated every cycle, according to Eq. 6, to ensure that a task does not remain unassigned indefinitely.

$$S_t = S_t + \delta_S \tag{6}$$

This update rule ensures that older requests have higher priorities since they will emit stronger stimuli, but also allow more specialized factories to bid for other types of tasks if necessary. As a note, different factors could be easily added to Eq. 6 to create other priority criteria instead of “unassigned time” (e.g. “unit priority”).

## Evaluation

The evaluation of our scenario consisted in running automated scripts that, at certain time intervals, requested one or more units at certain points of the map, simulating a typical human player input. To execute the orders, 3 factories were positioned close to each other, simulating a “base camp”. The type of the factories was chosen randomly, with the constraint that the production of both Footman and Rifleman units should be possible within this base camp. Each run was evaluated with 5 different base camp configurations.

To evaluate our approach, five different variants of our base scenario were implemented and tested within the Warcraft III game environment, ensuring that factors such as pathfinding, collisions between units and with the environment, were accounted for in the in-game simulation:

**Even (unit distribution):** at regular time intervals, a new request is issued. The requested unit has 50% chance of being either a Footman or a Rifleman.

**Uneven (unit distribution):** a new request is issued at regular time intervals. With 90% chance, the requested unit is a Footman, otherwise it is a Rifleman. When half of the units have been requested, the distribution is mirrored, i.e. there is 10% chance of the request being a Footman and 90% chance of being a Rifleman.

**Destruct (and construct):** in this variant, during the course of the scenario, two factories are destroyed and, afterwards, two random others (under the same constraints) are constructed.

**Burst:** instead of being ordered regularly, units are requested in bursts, i.e. in small time intervals, with longer pauses between bursts.

**More (factories):** the number of available factories is increased to 9 in this variant, grouped into 3 base camps of 3 factories, and spread out throughout the map.

To evaluate the performance of our approach, we ran each scenario variant using three additional algorithms:

**Random:** this algorithm randomly selects one of the available factories capable of producing the requested unit and having free slots in its production queue. This scenario serves primarily the purpose of creating a baseline.

**Closest:** this algorithm attributes production requests to the closest factory with empty slots in its queue. This algorithm simulates the assignment strategy commonly used by “newcomer” players.

**Global:** this algorithm attributes production requests to the expected fastest factory, taking into consideration the time needed for the requested unit to move from the factory to the target rally point in a straight line, and all production and setup times of the requests in its queue, plus the one being disputed. While not ensuring optimal performance, this algorithm takes into account the global state of the whole game at the time of the decision.

For evaluation purposes, the following data was gathered during each run:

**Unassigned time:** the time interval between the moment a unit is requested and the moment it is assigned to a specific factory (unsgnd);

**Queued time:** the time interval between the moment a request is assigned to a factory and the moment the factory starts its production (queued);

**Training time:** the time spent by the factory to produce the unit (traing);

**Delivery time:** the time a unit, after been produced, takes to reach the designated target rally point (delvry);

**Number of setups:** the total number of setups during the whole scenario (setups);

**Execution time:** the total scenario duration, that is, until the last unit reaches its target rally point (test).

The “closest” and “global” algorithms are deterministic and were executed 3 times for each script, to account for the stochastic nature of the game environment. Both “random” and WAIST are intrinsically stochastic, and as such were executed 15 times for each scenario. In both cases, the final result is the average of all runs.

Our approach requires setting a considerable amount of parameters. To specify and tune this set of parameters, we performed several iterations of smaller (and faster) runs consisting of 40 unit requests over the base scenario. After the parameters were defined, we performed the final tests which included 100 requests for each one of the five scenario variants and each one of the four algorithms, each run being evaluated with the five base camp configurations.

## Results

### “Even” Scenario

Fig. 3 shows the results for the “even” scenario. In this scenario, WAIST marginally outperformed “global”. Although WAIST takes longer to assign requests, this delays pays off since it significantly reduces the number of setups required (in this case, by 75%). Without response thresholds or the capacity to anticipate future requests, “global” execution time is penalized by the delays introduced by the setups. “Closest” was unable to share the workload between the different factories, since request assignment is exclusively based on distance. The lower “delivery” time did not compensate for this feature and set “closest” near the performance of “random”.

| | Unsgnd | Queued | Traing | Delvry | Σ | Test | Setups |
|---|---|---|---|---|---|---|---|
| Random | 19.94 | 87.85 | 21.80 | 8.75 | 138.36 | 697.60 | 16.73 |
| Closest | 16.15 | 91.01 | 20.73 | 8.07 | 135.98 | 695.66 | 15.33 |
| Global | 4.38 | 75.45 | 20.14 | 8.60 | 108.58 | 663.66 | 13.33 |
| WAIST | 9.49 | 69.14 | 19.83 | 8.55 | 107.02 | 661.93 | 3.40 |

Figure 3: Results for the “even” scenario.

## "Uneven" Scenario

Fig. 4 shows the results for the "uneven" scenario. In this scenario, "global" yielded better results. Because of the peculiar distribution of the request for both types of units, the number of setups for "global" was lower, and WAIST required a certain period of adaptation for the factories to loose their specialization, which in this case penalized the approach. Still, WAIST performed above both "random" and "closest" in this scenario, and only 1.5% below "global".

| | Unsgnd | Queued | Traing | Delvry | Σ | Test | Setups |
|---|---|---|---|---|---|---|---|
| Random | 23.06 | 81.75 | 20.78 | 9.55 | 135.16 | 833.06 | 10.00 |
| Closest | 22.35 | 80.33 | 20.28 | 8.69 | 131.66 | 820.00 | 6.33 |
| Global | 15.09 | 69.06 | 20.11 | 9.36 | 113.63 | 782.66 | 6.00 |
| WAIST | 20.03 | 71.43 | 20.07 | 9.31 | 120.86 | 794.60 | 2.93 |

Figure 4: Results for the "uneven" scenario.

## "Destruct" Scenario

Fig. 5 shows the results for the "destruct" scenario. In this scenario which included the destruction and construction of new factories while requests were issued, WAIST showed its robustness by achieving the best performance. Results suggest the reduced number of setups to be one of the main explanation for the fact. Peculiarly, "closest" had the highest "delivery" time in this scenario, due to the unexpected destruction of the factories closest to the rally point at the time of request assignment.

| | Unsgnd | Queued | Traing | Delvry | Σ | Test | Setups |
|---|---|---|---|---|---|---|---|
| Random | 21.27 | 92.62 | 21.68 | 9.82 | 145.42 | 729.20 | 22.00 |
| Closest | 17.47 | 87.09 | 21.32 | 9.92 | 135.81 | 712.00 | 23.06 |
| Global | 7.90 | 76.27 | 20.50 | 9.27 | 113.96 | 684.86 | 19.06 |
| WAIST | 11.15 | 75.33 | 20.32 | 9.26 | 116.08 | 676.66 | 2.86 |

Figure 5: Results for the "destruct" scenario.

## "Burst" Scenario

Fig. 6 shows the results for the "burst" scenario. In this scenario, in which units were requested by bursts rather than uniformly over time, "global" achieved the best performance, while both WAIST and "closest" shared a similar performance. This scenario demonstrated a possible weakness in the WAIST approach. Although this result could be related to the non-optimal tuning of $\delta_3$ and $\delta_4$, that control how fast a factory looses its specialization when idle, WAIST was unable to distribute work efficiently during the in-between-burst time intervals.

| | Unsgnd | Queued | Traing | Delvry | Σ | Test | Setups |
|---|---|---|---|---|---|---|---|
| Random | 49.18 | 106.92 | 20.72 | 13.16 | 189.99 | 691.26 | 13.33 |
| Closest | 47.24 | 103.61 | 20.58 | 8.51 | 179.96 | 679.33 | 12.33 |
| Global | 40.50 | 102.76 | 20.42 | 8.80 | 172.50 | 663.00 | 10.66 |
| WAIST | 46.95 | 103.75 | 20.50 | 8.86 | 180.07 | 678.20 | 9.00 |

Figure 6: Results for the "burst" scenario.

## "More" Scenario

Fig. 7 shows the results for the "more" factories scenario. In this scenario, "global" outperformed WAIST by only 3%, while ahead of both "random" and "closest". Unexpectedly, WAIST did use approximately the same number of setups as "global". A possible explanation for this fact is that, as more factories are available, idle periods are longer and several factories may experience a premature loss of specialization. As in the previous scenario, the rate at which the response threshold decreases may not be the more adequate for such scenario and fine tuning the related parameters may provide better results.

| | Unsgnd | Queued | Traing | Delvry | Σ | Test | Setups |
|---|---|---|---|---|---|---|---|
| Random | 1.00 | 12.53 | 22.13 | 13.15 | 48.81 | 545.86 | 28.26 |
| Closest | 1.00 | 55.34 | 22.42 | 5.45 | 84.23 | 615.66 | 27.66 |
| Global | 1.00 | 1.92 | 20.12 | 7.37 | 30.43 | 492.66 | 15.66 |
| WAIST | 1.04 | 3.08 | 21.01 | 12.46 | 37.60 | 507.66 | 16.06 |

Figure 7: Results for the "more" scenario.

An important remark is that, due to the high number of available factories in comparison to the request rate, "random", "closest" and "global" assigned production tasks to factories without delay. This could be an indication that WAIST is more indicated when congestion is experienced, although its performance, while not optimum, is acceptable in less congested situations.

## WAIST Overall Performance

Fig. 8 compares the total execution time for each algorithm across the five different scenario variants used during the evaluation phase. Both WAIST and "global" algorithms achieved a similar performance, "global" only outperforming WAIST by an average of 0.9% over the five scenarios.

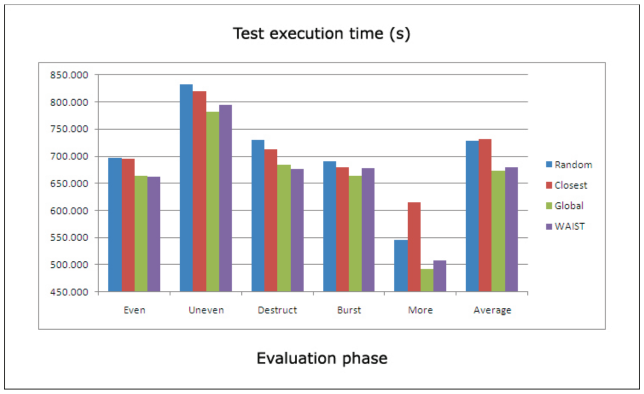

Figure 8: Overall results of the evaluation phase.

We consider this an encouraging result for WAIST, taking into consideration that scheduling relies only on local information, while "global" has access to all information at once. In comparison to "global", WAIST presented the additional advantage of allowing factories to wait for best suited requests by using the response threshold mechanism.

In this work, WAIST parameters were tuned by hand. We believe the use of an optimization algorithm could provide a better set of parameters, especially when the specific characteristic of the scenario are known beforehand, although its use would require a considerable amount of time when considering the simulation running in an environment such as the Warcraft III game engine. In future work, we intend to research whether the performance of WAIST would further increase using optimal tuning of the parameters for each specific scenario variant.

### WAIST Benefits

Although not the most efficient approach in all possible situations, WAIST demonstrated to be an efficient and versatile approach for unit production scheduling in RTS games, even when compared to a centralized algorithm. WAIST was found to be most appropriate in games with the following characteristics:

- Significant setup times: in this work, setups accounted for approximately 10% of the time needed to train or produce a unit. We expect higher setup times to boost WAIST performance in comparison to the other approaches;
- Congestion of requests: WAIST is most suited for situation with continuous high request rate, which is expected at higher levels of play. Due to the lack of idle times, specialization ensures the adequacy of the algorithm;
- Performance scalability: WAIST is adequate for larger scale scenarios. While the production queue and the capability of all factories are evaluated to select the best possible choice in a centralized solution, bidding and dominance contests require less computation power while achieving a similar performance.

Another benefit comes from its agent-based nature. In this work, due to the use of the Warcraft III World Editor, the algorithm is executed sequentially. However, this algorithm can take advantage of the parallel computation offered by multi-core processors and multi-agent systems, increasing its computational efficiency even further.

## Conclusion

In this paper, we presented a new paradigm for scheduling unit production and training in real-time strategy games, inspired by the social behaviour of wasps which, using only local knowledge and cues, are able to efficiently allocate tasks while adapting themselves to the inconstant colony needs and frequent environment changes. We adapted the R-Wasp algorithm, a wasp-based task allocation algorithm applied to certain engineering problems, to the real-time scheduling of units in a real-time strategy game, and described WAIST, a first step toward the exploration of new paradigms of gameplay for real-time strategy games relying on the support of player-friendly artificial intelligence.

We described a series of five scenarios with which we evaluated WAIST under different situations within the game environment of the game Warcraft III. The scenarios introduced a new feature observable in real factories to the current language of game mechanics related to real-time strategy games: heterogeneity i.e. "factories" with different characteristics, and extra production costs, either of resources or time, resultant from the change of production. To get a better idea of the benefits and limitations of the approach, we compared WAIST performance in each of the five scenarios to three other approaches: random attribution; distance-based attribution, and global attribution which considers all the information available in the game environment.

Overall, WAIST performed comparably to the global attribution algorithm (and better than the other two), an encouraging result considering WAIST is a decentralized algorithm that relies on local information while the latter has full global knowledge. As such, we believe WAIST to be an efficient and reliable alternative for real time scheduling in real time strategy games. While WAIST experiences some limitations when dealing with low amounts of requests, it demonstrated good performance in situation of higher congestion of requests, and when setting up from one production to another is expensive. The limitations observed in these situations could be related to fine tuning specialization-related parameters for the specificities of each scenario, and should be researched further. Due to its decentralized and multi-agent nature, WAIST should also be able to scale relatively well performance-wise, and take advantage of multi-core processors.

## References

Beni, G., and Wang, J. 1989. Swarm intelligence in cellular robotic systems. In *Proc. NATO Advanced Workshop on Robots and Biological Systems*, 26–30.

Cicirello, V., and Smith, S. F. 2001. Wasp nests for self-configurable factories. In Müller, J. P.; Andre, E.; Sen, S.; and Frasson, C., eds., *Proceedings of the Fifth International Conference on Autonomous Agents*, 473–480. ACM Press.

Engelbrecht, A. 2006. *Fundamentals of Computational Swarm Intelligence*. Wiley.

Morley, D. 1996. Painting trucks at general motors: The effectiveness of a complexity-based approach. *Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business, The Ernst and Young Center for Business Innovation* 53–58.

Perna, A.; Jost, C.; Valverde, S.; Gautrais, J.; Theraulaz, G.; and Kuntz, P. 2008. The topological fortress of termites. In *Bio-Inspired Computing and Communication*, 165–173.

Sendova-Franks, A.; Scholes, S.; Franks, N.; and Melhuish, C. 2004. Brood sorting by ants: two phases and differential diffusion. *Animal Behaviour* 68.

Theraulaz, G.; Goss, S.; Gervet, J.; and Deneubourg, J.-L. 1991. Task differentiation in polistes wasp colonies: a model for self-organizing groups of robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 346–355.

Traniello, J. F. A. 1989. Foraging strategies of ants. *Annual Review of Entomology* 34:191–210.