Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 1331–1336
DOI: 10.15439/2014F357
ACSIS, Vol. 2

Adaptive scheduling in dynamic environments

Hanno Hildmann, Miquel Martin
NEC Laboratories Europe
Kurfürsten-Anlage 36
D-69115 Heidelberg
Email: {hanno.hildmann, miquel.martin}@neclab.eu

***Abstract*—We present a method for fair agent scheduling in transportation scenarios. The approach is designed to first ensure the scheduling of all required task locations and, once this is achieved, focus on a balancing the workload across the population of transportation units. This, while almost certainly sub-optimal in the context of efficiency, facilitates the speedy allocation of new geographically located tasks due to the distribution of the remaining capacity across the agent population.**

We discuss our method, present results from simulations and discuss the advantages and disadvantages of the approach.

I. Introduction

LARGE scale transportation scheduling is a well known high complexity problem [1]. Arguably, the best known instance is the class of *Travelling Salesman Problems*, of which there are many variations which have been extensively discussed in the literature. Generally speaking, the problem is known to be NP-complete, meaning that while it is straight forward to check whether a given solution is correct, it is extremely difficult to construct such a solution [2].

In the classic *Travelling Salesman Problems* (TSP) one is concerned with finding the shortest path that connects a finite set of locations; in the multiple travelling salesman problem (MTSP) this is extended to a finite set of disjoint / mutually exclusive routes. The method presented in this paper solves a problem similar to the MTSP with the difference that we are not interested in the shortest path but in a collection of paths that do not exceed a certain length or agent capacity.

A. Motivation

The motivation for this is the idea that in the considered use cases we are in charge of a fleet of mobile agents which have a certain capacity (fuel, battery level, working time, etc) which we deem an acceptable investment.

While it is of course of interest to reduce the aggregated amount of the consumed resources we focus instead on the quick adaptation we can offer in a dynamic environment. Specifically, the empirical results we present are generated using a simulation that adds tasks to the problem after the initial solutions have been created. The motivation for this is that we consider scheduling scenarios where locations may be added throughout the active period of the fleet.

B. Aim of the paper

The aim of this paper is to present the method used and prove its performance through empirical results. The evaluation is intentionally kept generic and the focus is on providing the reader with all the information required to apply the method to specific problem instances. To this end, a variety of parameters are reported without any claim regarding their optimal settings (since we present results on a generic simulation it is of little interest to report on tuning results).

C. Application scenarios

The initial use case for the approach is the dispatch of service personnel to a number of locations, as major telecom operators or internet providers would routinely do. This can be extended to maintenance personnel as well as corporate security services on large estates. It should be noted that neither time windows, nor ordering locations according to some priority are currently supported.

We are at the moment considering the approach for allocating computational tasks to processing resources (e.g. servers in a server farm), also, the approach has potential in large scale disaster relief or humanitarian aid scenarios.

II. Method

We present a novel method [3] for adaptive scheduling in dynamic environments. At the core of the approach is a self determined paradigm shift [4] which enables a population of agents to switch their priorities on the basis of their (locally) available information: agents are assumed to have a limited view on the tasks, meaning that they are only aware of tasks within a certain distance from their position or their path. Their decisions are based on this limited partial visibility [5].

By *stance* we understand a predisposition to act one way or the other [6]. Specifically, agents are considered to be *maximizing* (which we will abbreviate to *max*) if they are aware of un-scheduled tasks, and *balanced* (*bal*) otherwise.

During each iteration all agents are triggered to locate a task within their reach (i.e. within their remaining capacity) and to attempt to assimilate this task into their own schedule. They will make this decision stochastically and depending on their and the other agent's remaining capacity. The other agent here is the agent to whom the task is currently scheduled, if there is no such agent the decision is foregone and the agent will simply schedule the task (cf. Figure 1). If the task is already scheduled to an agent then a value is calculated for both agents and from these a probability for re-allocating the task is derived.

978-83-60810-58-3/$25.00

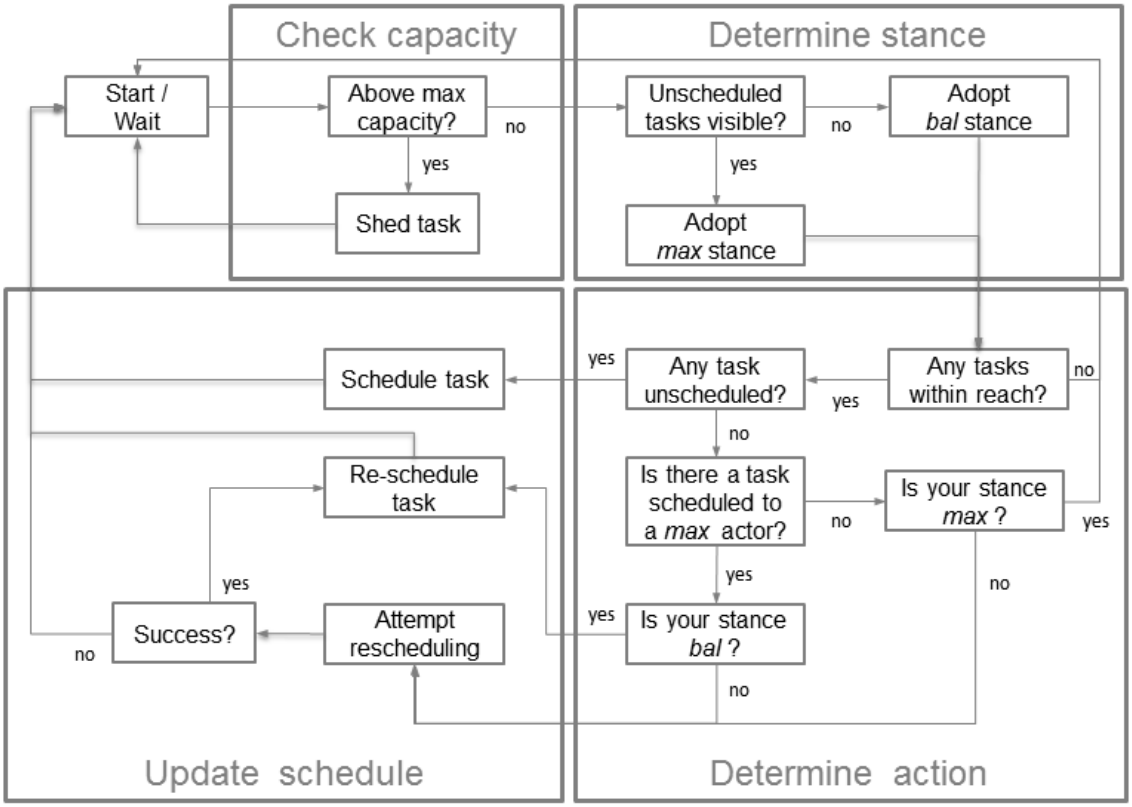

Fig. 1. Flow diagram of the approach

By reversing this value (the probability of successfully assimilating a task) we enable the agents to shift between two polar opposite stances: *rich gets richer* and *rich gets poorer*, where richness is associated with load (i.e. the higher the load the *richer* the agent). While the *max* stance means that an initially homogenous population will split into two groups: one that is maximised its load and one that is minimizing it, the *bal* stance results in agents with a higher load constantly losing tasks to agents with a lower load, effectively balancing their loads between themselves and their neighbours.

We require the approach to be non-deterministic (through the use of random elements like. e.g. the order in which agents are triggered, the choice of which task to consider for re-allocation and the stochastic decision whether they will win or lose the bid for re-allocating a task), as well as to be applied many times over. This will not result in an immediate convergence towards the best possible load distribution, but instead will slowly converge towards a *fair* state, i.e. the load distribution that results in roughly even loads for all agents.

Given that the choice of which task to consider is non-deterministic (and thus not related to the stance of either agent) we have 4 possible situations to consider (2 stances for each of 2 agents) when defining the interaction between any two agents. For the remainder of the paper we will call the initiating agent *active* and consequently call the other agent (the agent to whom the task in question is scheduled) *passive*. These 4 combinations and the resulting interactions are:

- **bal-bal** (*active* agent *balanced* - *passive* agent *balanced*): Agents that are not aware of unallocated tasks pursue their long term objective (load balancing). The agents will follow the *rich gets poorer* paradigm when calculating whether to reschedule the task.
- **max-max** (*active* agent *max* - *passive* agent *max*): If both agents are aware of un-allocated tasks they will focus on their short term objective, which is to quickly assimilate new/unscheduled tasks. They will follow the *rich gets richer* paradigm which will result in some agents depleting their capacity almost entirely while others are freed up, enabling them to assimilate the new tasks.
- **max-bal** (*active* agent *max* - *passive* agent *balanced*): If an agent is aware of unallocated tasks it is not interested to receive any tasks from an agent that is not. In this case, the interaction is aborted (this is simply prevented by restricting the choice of tasks, and thus agents).
- **bal-max** (*active* agent *balanced* - *passive* agent *max*): Balanced agents that happen to come across a task currently scheduled to an agent that is aware of unallocated tasks will always take the task in question. The reasoning is that the overall aim is to remove as many tasks as possible from the agents that are struggling to accommodate unallocated tasks.

III. Model

A. Parameters

The following are the parameters used for the model:

- number of agents
- number of tasks
- capacity (the same for all agents per simulation run)
- visibility range (considerably smaller than capacity)
- map size (this was 100x100 for all simulations)
- single depot versus individual depots

In line with our intention to enable the reader to implement and evaluate the approach independently, we also briefly discuss the only tuning parameter which we have included in this paper: α (used in the formulae in §III-B). Changes in α will affect the speed with which the approach converges to a somewhat stable solution, while on the other hand determining the degree of change in the environment which the algorithm can handle while still performing well.

B. Formulae

The decision whether to re-allocate a task from the active agent to the passive agent is stochastic. The probability of a *max* agent stealing as task from another *max* agent is:

$$P_A^{max} = \frac{(rem.capacity_B)^\alpha}{(rem.capacity_A)^\alpha + (rem.capacity_B)^\alpha} \quad (1)$$

with P_A^{bal} the probability of agent *A* stealing a task from *B*, and $rem.capacity_X$ the remaining capacity of an agent *X*.

The corresponding P_A^{bal} is simply the opposite of P_A^{max}:

$$P_A^{bal} = 1 - P_A^{max} \quad (2)$$

In other words, $P_A^{bal} = P_B^{max}$, which preserves the symmetry of the probabilities in the opposing stances, i.e. if *A* were likely to win for *rich gets richer*, it should be equally likely to loose for *rich gets poorer*.

C. Scenario

a) Agents: have a capacity, a visibility range, a starting depot and a route. The capacity is static, while the *remaining capacity* is the capacity minus the cost of the current route.

b) Tasks: have a location, expressed by x-y coordinates.

c) Depots: mark the beginning and the end of each route. Each agent is assigned a depot, expressed by x-y coordinates.

d) Routes: start and end with the depot of the respective agent. Routes are assigned a cost, which is the sum of the travel distances between the individual entries in the route. The distance is calculated as the Euclidean Distance.

e) World: has a map, a list of agents and a list of routes (schedules). The act of re-allocating a task from one route to another is assumed to be instantaneous and cost free.

IV. Simulations

A. Scenarios

Two scenarios were used for the simulations:

1) 40 tasks were added to the map once (at iteration 25): This is used for smaller scenarios, where the 40 tasks constitute a substantial percentage of the existing tasks, and where a small number of agents attempt to efficiently schedule the new tasks. This scenario is used to see whether, and how fast, the new tasks can be allocated and how long it takes for the average of the individual schedules to converge to a somewhat stable value.
   All tasks were randomly given coordinates in the range ($[50, 70]$, $[-50, 50]$); since the board is initialized to $[-50, 50]$ for both x and y coordinates these new tasks appear across the range of the y axis and further on the x than any other tasks.
2) 100 tasks are added every 25 iterations:
   This scenario is used to investigate the robustness of the approach and to see how a simulated population holds up in the face of a task load that is getting closer and closer to the total capacity of the population. Contrary to the above these tasks were placed all over the map ($[-50, 50]$, $[-50, 50]$).

B. Route calculations

We are interested in the least cost for a schedule, that is, the shortest path thorough all tasks in a list. This is the well known *Travelling Salesman Problem*, which is known to be NP-complete. Since the algorithm will need to calculate this for every task in the vicinity of an agent when computing the list of visible or reachable tasks, this is a calculation that is performed many times (millions of times, to be more precise, for any of the presented simulations).

To reduce the computation times two steps are taken: First of all, for routes of short length ($\leq$8) the best route is calculated via brute force, as this has proven to be the most effective approach for us. Secondly, for longer routes, we use simulated annealing with very soft convergence criteria. Because paths are recalculated and built upon at every iteration, repeatedly accepting a sub-optimal path still tends to progressively improve the solutions, yielding good results.

C. Extremely large scenarios

We investigated the performance of the approach for very large scenarios to verify that the approach is a) scalable and b) does indeed perform better with larger agent populations. To this end we simulated a scenario with 10,000 initial tasks and 1000 agents. We then added 100 tasks every 25 iterations and let the simulation run for 1000 iterations. This was done 4 times with different seeds. The generation of the results took 4-5 weeks because of the massive number of route calculations that had to be done.

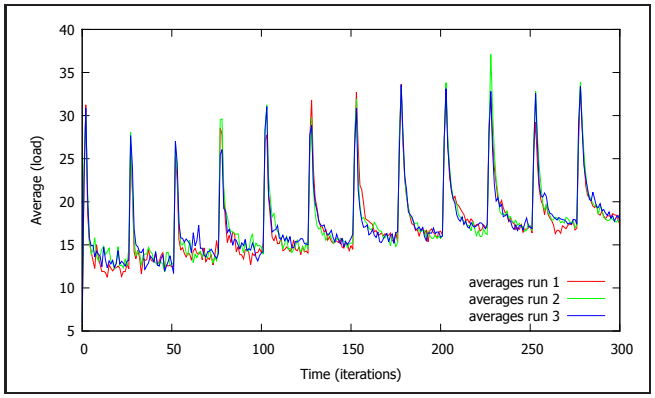

Fig. 2. Three separate runs of the simulation (x-axis: iterations; y-axis: load). Simulation setting: Scenario 2, agents = 50, tasks = 100, capacity = 75, 300 iterations, visibility = 40, map = 100x100. Δ: 100 new tasks every 25 steps

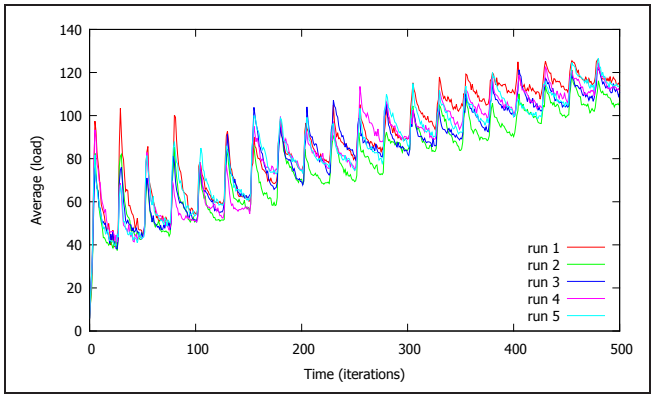

Fig. 3. Five separate runs of the simulation (x-axis: iterations; y-axis: load). Simulation setting: Scenario 2, agents = 50, tasks = 100, capacity = 150, 500 iterations, visibility = 40, map = 100x100. Δ: 100 new tasks every 25 steps

V. Results and Discussion

A. Parameter space exploration

We ran a number of identical simulations (for each agent population size, simulations were ran multiple times with different seeds, the seeds used for the different population sizes were the same so as to ensure comparability of the results) where we increased the number of agents to investigate the relevance of the size of the agent population. To this end, Scenario 1 was used and separate batches were run for small populations (20-100 agents, increasing in steps of 10) and large populations (200-1000 agents, increasing in steps of 100). The adaptation of the new tasks worked equally well across all population sizes, with tasks being assimilated into schedules as fast as possible (considering that each agent can only add one task per iteration; larger populations could assimilate tasks faster). The same holds for the aggregated load as well as the individual agents' loads. Other than the obvious differences in the performance, the algorithm performed well, even for very small numbers of agents (e.g. 20 agents for 500 tasks).

B. Performance evaluation

1) Adaptation of new tasks: To investigate the adaption of new tasks into schedules we looked at the average load of the *max* agents (as their the presence indicates the existence of unallocated tasks). For very small numbers of agents (fewer than the number of new tasks that were added) there is a noticeable difference, but as soon as the number of agents equals the number of new tasks there is no more difference in the number of turns it takes for all new tasks to be assimilated.

2) Optimization of the aggregated load: Addition of new tasks accounts for a sudden increase in the averages. For all but the smallest agent populations, however, we can see a steady and fast decrease in the reported averages. As expected, smaller populations struggle more with the process of decreasing the aggregated workload. This can be explained by the fact that for smaller populations the exchange of tasks is much more likely to have a more dramatic impact on their route length. This was also reflected in the standard deviations.

3) Optimization of the individual agents' loads: Population size has an impact on the standard deviation itself as well as on the rate of its decline. Smaller populations seem to quickly decrease the standard deviation between time step 120 and 140. This was, however, not reflected in the averages themselves. For larger populations we witnessed a much smoother return to small standard deviations, with the largest populations reaching a plateau very quickly.

4) Adaptivity and resilience: In order to evaluate the resilience of the approach and to investigate how the approach performs when it subjected to repetitive heavy strain, we ran a number of concurrent simulations for 300, 500 and (due to time constraints) one single simulation for 1000 time steps. All simulations started with 100 initial tasks which were supplemented by another 100 every 25 iterations thereafter (Scenario 2). The first set of simulations increased the number of tasks to 1200, the second set to 2000 and the final simulation ended with 4000 tasks all-together. While the first of these 3 batches was run with a capacity of 75, the latter two used a capacity of 150 (so as to provide the agents with the capacity required to accommodate all the new tasks). As before, we investigated the performance with regard to the aggregated load of all agents as well as the load of the individual agents:

a) Aggregated load of the population: Figures 2, 3 and 4 show the average loads of the agents over the course of the simulations. Even for large (and very large, as depicted in Figure 7) numbers of added tasks, the rate of convergence towards a good average remains stable. The overall increase of the load is not very large, and is decreasing with time as new tasks are added. This is understandable since the agents are using the shortest path through all tasks to calculate their load, and for increasing task numbers new tasks are more likely to be very close to already scheduled tasks.

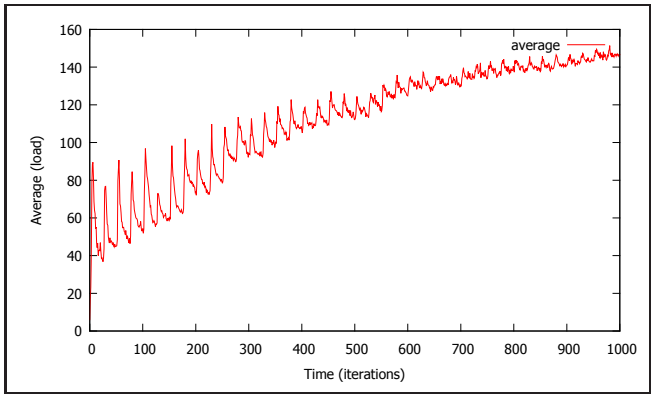

Fig. 4. The average load (y-axis) for one long simulation (x-axis: iterations). Simulation setting: Scenario 2, agents = 50, tasks = 100, capacity = 150, 1000 iterations, visibility=40, map=100x100. Δ: 100 new tasks every 25 steps

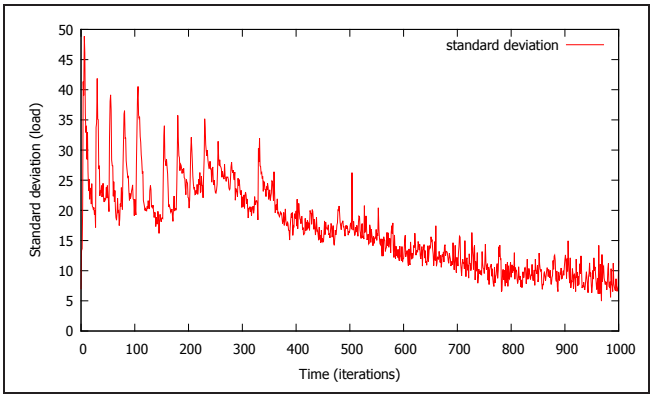

Fig. 6. The standard deviation (y-axis) from the average load for agents over 1000 iterations (x-axis). Setting: Scenario 2, agents = 50, tasks = 100, capacity = 150, visibility = 40, map = 100x100. Δ: 100 new tasks every 25 steps

b) Performance of max agents: Regarding the performance of the *max* agents, we can see in Figure 5 that each time new tasks are added there is a quick burst of activity before the averages drop again sharply. This corresponds to the number of *max* agents dropping to zero and the balanced agents optimizing their schedules. Besides showing us that the time used for the assimilation of the new tasks does not increase with larger task counts, it also demonstrates the effectiveness of the paradigm shift. These claims are best discussed in combination with the next paragraph.

c) Load balancing between individual agents: Regarding the performance of the individual agents, the graphs presented in Figure 6 show the standard deviation from the averages (already discussed above). We can see that the deviation from the average more than doubles briefly as the balanced agents take on tasks from the *max* agents. However, in all simulations it then drops almost immediately to the previous values. There seems to be a turning point around time step 200 (or when the task count has reached 900) after which the deviation is slowly decreasing. This is to be expected: with an increasingly denser task distribution we can realistically assume that the individual agents can balance their load more minutely.

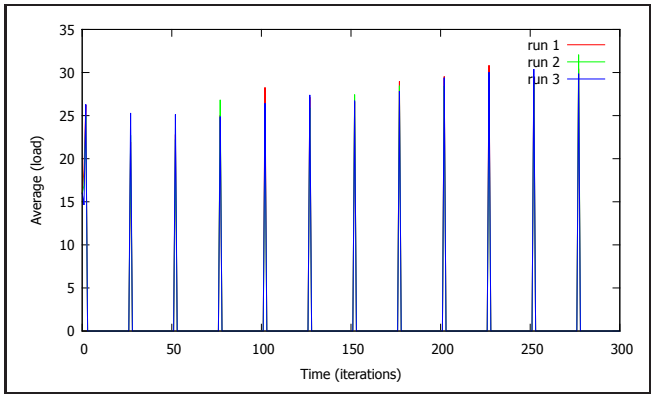

Fig. 5. The averages (y-axis) of the *max* agents for 3 separate runs (x-axis: iterations). Setting: Scenario 2, agents = 50, tasks = 100, capacity = 75, 300 iterations, visibility = 40, map = 100x100. Δ: 100 new tasks every 25 steps

d) Performance of balanced agents: We observed that the averages of the balanced agents jumped each time a new batch of tasks was added to the map but quickly dropped again, with the standard deviation following shortly thereafter.

Figures 4 and 6 report on results from small agent populations while Figures 7 and 8 show the performance of very large populations, both running for 1000 iterations, but with widely different task counts.

In Figure 4, the averages of the balanced agents increased slowly for smaller agent populations (capacity = 150) but their averages converged towards a plateau at around 120. The standard deviation, converging towards 25 (which is about the remaining capacity left for the agents) explains this plateau.

Regarding the corresponding standard deviation (Figure 6), we noticed a decrease in the deviation with increasing task counts. This matches the increased averages (decreased remaining capacity) but can also be explained by the fact that in larger task count new tasks are placed in closer proximity of existing (and already scheduled) tasks. In addition, the load balancing improves with the number of tasks to exchange.

As for the much larger simulation reported upon in Figures 7 and 8, we argue that the results show that the approach is scalable. We furthermore suggest that the increases in average load and standard deviation are stable because, unlike the other simulation discussed above, the agents had sizable remaining capacity and the sheer number of the agents in the simulation resulted in a much higher chance of tasks being placed in close proximity of agents.

The patterns observed in the other simulations are repeated here: after tasks are added a brief period of ensues within which the agents change paradigm, assimilate the new tasks and increase their load. This is followed immediately by a steady optimization which reduces the average load to (almost) the values from before the adding of the tasks. As far as the standard deviation is concerned, while the above graph (Figure 6) shows a convergence towards somewhat stable values, the much larger simulation (Figure 8) does not. This is because in the larger scenario agents can still assimilate far away tasks.

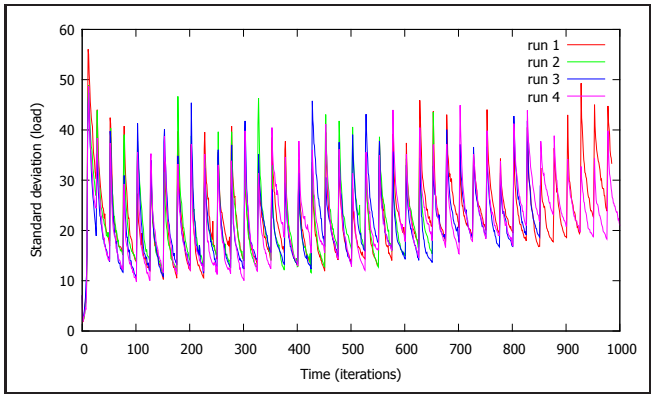

Fig. 8. The standard deviation (y-axis) from the average load for 4 runs (x-axis: iterations) of a very large simulation (Scenario 2, with 1000 agents) starting with 10,000 tasks and increasing to 14,000 tasks. Setting: see Fig. 7

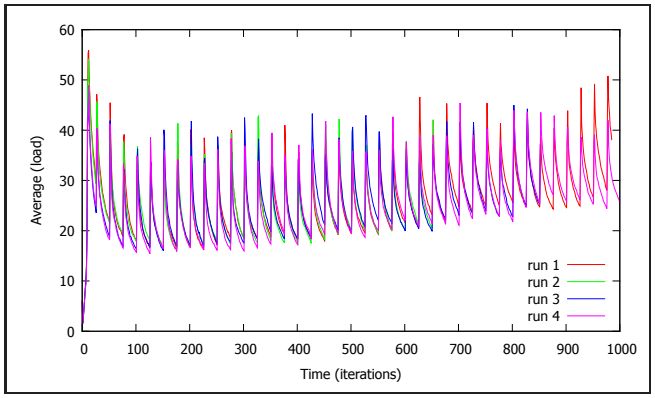

Fig. 7. The average load (y-axis) of the agents for 4 runs (x-axis: iterations) of a very large simulation (Scenario 2, with 1000 agents) that started with 10,000 tasks and over the course of 1000 iterations increased to 14,000 tasks. Setting: capacity = 400, visibility = 40, map = 100x100. Cf. also Fig. 8

C. Restrictions to the approach

The proposed method relies on a number of agents working in proximity, such that the schedule assigned to a specific agent can be partly absorbed into another agent's schedule. It is expected that there is a critical mass which is required in order for the method to outperform other approaches. We tested the approach against small numbers (as few as 20) of agents and it performed well; however the critical mass depends on the size of the map, the visibility and the capacity of the agents.

Furthermore, there is an upper limit to the degree of change over time in which the method can be expected to perform well. If the changes are too rapid or dramatic, recalculating the entire solution will be the better approach. This is due to the iterative nature of the approach, which will quickly adapt to changes and follow moving centers of gravity in the problem space. If such centers appear and disappear seemingly at random it becomes impossible to follow them, and thus the method loses its edge over other approaches.

D. Closing remarks

The combination of probabilistic decision making as well as the balancing and maximizing paradigm results in the minimization of agents to which a subset of the tasks (i.e. tasks from some region of the problem space) is scheduled to. This means that under the right conditions some agent might have very few or no tasks scheduled, which enables them to take on tasks from other regions of the problem space. This is expected to result in the ability to cross over large distances and effectively to reallocate agents within a problem space. Local optimization techniques normally consider only local alterations to a solution and recalculating the complete set of schedules for the whole problem space may be expensive in terms of computational effort.

REFERENCES

[1] R. Xu and I. Wunsch, D., "Survey of clustering algorithms," *Neural Networks, IEEE Transactions on*, vol. 16, no. 3, pp. 645 –678, May 2005.

[2] H. Hassan and A. Al-Hamadi, "On comparative evaluation of Thorndike's psycho-learning experimental work versus an optimal swarm intelligent system," in *Computational Intelligence for Modelling Control Automation, 2008 International Conference on*, Dec. 2008, pp. 1083 –1088.

[3] Patent, "Ditributed task scheduling using multiple agent paradigms," US Patent No: 14/250,470 (filed), H. Hildmann and M. Martin (inventors), Mar. 2014.

[4] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-Organization in Biological Systems*. Princeton Univ Press., 2001.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for optimization from social insect behaviour," *Nature*, July 2000. [Online]. Available: http://dx.doi.org/10.1038/35017500

[6] D. M. Gordon, "The organization of work in social insect colonies," *Nature*, vol. 380, pp. 121–124, Mar. 1996. [Online]. Available: http://dx.doi.org/10.1038/380121a0