U N I V E R S I T Y O F T A R T U
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Riivo Roose

Automated Resource Optimization in Business Processes

Master's Thesis

Supervisor: Prof. Marlon Dumas

Author: ..... May „.....„ 2012
Supervisor: ..... May „.....„ 2012

Approved for defense:
Professor: ..... May „.....„ 2012

TARTU 2012

Contents

1. Introduction

Business process management is the process of modifying or adjusting an organizations business process in order to achieve higher productivity or lower costs. Each company or organization has a value creating process that usually involves people, machines and information. One of the main problems with such processes is that it is very difficult to predict how much of each resource is actually needed. This is especially important in cases where business requirements and demand change very frequently. For example, in a factory producing ice cream the demands are significantly higher during the summer season. In processes where machines do most of the work optimization is simpler as one must just reduce the time the machines are running. On other hand there might be an insurance call centre where the optimization is more complex. It is very likely that people in different roles might do multiple tasks in the business process. As a result it is not very straightforward to predict the amount of workers needed. Furthermore, the amount of calls might differ between morning and evening; also there might be more calls during a storm season. One way to analyse such a call centre business process is to map it using some business process modelling language (BPMN/BPEL/..). These languages were introduced so that business processes could be easily described in a manner that all the stakeholders could understand. For example, BPMN can be an effective way to communicate between the financial side and IT as the diagrams are generally self-explanatory. After the process has been mapped into a modelling language it is possible to simulate it using some business process simulation tool like IBM WebSphere Business Modeler or Visual Paradigm Business Process Visual Architect. In order to do such simulation one has to define resource availability, process durations and gateway path selection probabilities. Some tools give more power to the user by giving them the possibility to create more realistic scenarios – for example one can set inputs to arrive in patches. Another useful feature is the possibility to have variance in the arrival rate or task durations. In a factory production line processes usually start with a predetermined interval but in any shop or service centre one has to be prepared that multiple customers arrive at the same time or very close to each other.

Running a business process as a simulation gives us a quantitative overview of how the process is performing or is predicted to perform under a given allocation of resources to tasks. The output of a simulation result typically includes average/total process duration, average/total waiting times, resource utilization and various types of costs. One of the most

useful features of running a simulation is that one can identify bottlenecks in the business process. If a task is causing delays in a process then its waiting times will be significantly higher than for other tasks. On the other hand, if there is a shortage of certain resources then the resource utilization of the given resource will be noticeably higher than for other resources. These two attributes are important if one is aiming to provide a better service for customers or improve the throughput of a factory. However, there are also some business processes where speed is not of the highest priority and this gives an opportunity to optimize on cost. Imagine a scenario where hiring one highly paid specialist instead of two might slow down the process by 25% but will save costs by 40%. In such situations process throughput or duration is sacrificed in most companies.

There are multiple approaches to optimizing a business process. One approach is to redesign the process model so that the tasks are done in a different order. The purpose of such optimization is to reduce the amount of tasks that get held up by process bottlenecks. Relocating a bottleneck task to a position which is reached by less process instances should improve overall performance. Another situation where relocating a task to a later stage might be useful is when one has a very parallel business process and some tasks need all of the branches to be finished before it can be executed. Leaving such tasks until the end of the process might have a significant impact on process throughput. In this spirit, [1] introduces a semi-automated method that uses process and operation data to identify bottlenecks in the system. With the help of the the *dBOP* platform explained in the paper the analyst is displayed with information as well as guidance on how to improve the process by relocating specific tasks.

Another approach for optimizing business processes is to leave the process model as it is but try out different amounts of resources, consider training the employees so that the task execution times are reduced or rearrange the resources which are responsible for executing a given task. One can always assume that increasing the amount of resources will improve the average cycle time of the process until the minimum cycle time is reached. Meanwhile there are a lot of cost considerations that have to be taken into account when deciding to train one's employees so that they would perform their tasks quicker. If the improvements in cycle time are not large enough they will not justify the training costs. Rearranging tasks between resources is another optimization technique. In this technique one task can be allocated from one resource to another. This might be useful when the training costs are low and the utilization of one resource is very low. However, caution should be taken

when reassigning tasks as it is not easy to predict whether the new assignees will manage to handle the additional workload. Another downside of such rearrangement is that the quality might suffer if people from one role have to start doing the job of other specialists.

In a perfect scenario all of the previously mentioned techniques should be used in order to optimize a business process. However, there exists a significant obstacle: business process optimization using simulation is by nature a very time-consuming activity. First, such optimizations involve a lot of manual trial-and-error work, where a business analyst thinks about the process and then tries to guess which modifications make the process faster, more reliable or cheaper. Such an approach does not guarantee to achieve better cost, cycle time or resource utilization. Even if the modifications do improve the process then the analyst is never guaranteed to achieve the best possible result. Second, most business process simulation software that is currently available is slow and simulation execution times are more likely to be counted in minutes than seconds on a standard desktop computer. These two reasons are likely to create a situation where a business analyst will be assigned a new task before he or she has fully analysed the process in hand. With such a manual approach it is still possible to achieve very good process optimization results but one is more likely to be far from it. Last year during a course project I came upon such a problem. I was trying to optimize the resource allocation for a rather complex business process that took around 15 minutes to run on IBM's WebSphere Business Modeler. As this performance was not acceptable I tried the same simulation on some competitive products but the simulation times were just as bad. In the end I had to submit a report where I had only tried around 15 different resource allocations for the given process. Although I managed to significantly improve the process durations I was still not happy about the results as in reality I had only tried a fraction of possible resource allocations. One solution to avoid such situations is to use a faster simulator. For example, a previous Master's thesis [2] led to a BPMN process simulator that significantly outperforms commercial simulation tools. This simulator (now called BIMP) supports most of BPMN notation and as a result should certainly be considered when attempting to simulate various process models. The speed of the tool might become useful when planning to redesign a business process or trying to reallocate different resources to new tasks. However, when one is trying to find the optimal resource allocation for a given process then purely just a fast simulator will be insufficient as the amount of possible resource allocation

combinations is usually very large and exploring these combinations manually in search for an optimal solution is a daunting task.

In light of the above, the objective of this thesis is to implement a business process optimizer that is capable of optimizing the allocation of resources to tasks in a given business process. In other words, the structure of the process (e.g. order of tasks) is assumed to be fixed and the focus is on determining the optimal number of resources that should be allocated to each *resource pool*. A resource pool is a collection of resources that can perform a given role or that have a certain capability required to perform certain tasks. The optimization will be done with respect to two quantitative key performance indicators (KPIs), specifically resource cost and cycle time.

Resource cost shows how much money is spent on resource wages. Resource cost can be calculated in multiple ways but the two main ones are the cost that is calculated based on the amount of hours worked (wage times resource utilization) and the cost based on the monthly wage of the resources allocated to tasks in the process, regardless of whether these resources are busy working on these tasks 100% of their time working or only a fraction of their time. The first approach to defining resource cost is useful when one is employing part-time workers to whom one only pays for the work they actually do. It is also useful in situations where one can deploy staff in other business processes but he/she wants to optimize the expenditure of the current process based on hours worked. The second approach is suitable in the case where the workers assigned to a task in a business process work full-time on the given process and do not get involved in any other processes. For the sake of scoping the work, this thesis focuses on optimizing the resource cost based on monthly wage. In particular, in the case study described in this thesis, workers are dedicated full-time to the process being optimized. The principles of the optimization solution developed in this thesis can still be applied to the optimization of resource cost defined as ‘wage times resource utilization’, but I do not explore this alternative in this thesis.

Cycle time is another useful feature of business processes. It represents the average execution time of process instances in one given simulation. Just like resource cost it is always preferable to have as low of an average cycle time as possible. Unfortunately, lower cycle time usually means hiring more people which raises costs. Another important feature of cycle time is that for some weekly business processes it is important that the cycle time is always below a certain point. For example, in a post office that receives 10 clients an

hour each client should be served with an average cycle time of 6 minutes. If cycle time is lower than 6 minutes then the post office will gain value in customer satisfaction. On the other hand, if the cycle time is above 6 minutes then it will start creating backlogs that will cause even larger delays in the subsequent opening hours.

Another relevant performance measure in the context of this thesis – although I do not seek to optimize along this dimension – is *resource utilization*. Resource utilization shows the percentage of time the worker is occupied from all the time he or she was available to do work. Generally one should aim for resource utilization close to 100% but in such cases the business process will become very prone to delays. If one resource is almost always occupied then there will be no buffer for cases where a task accidentally takes longer than expected or many customers arrive at the same time. Therefore resource utilization should be planned to be high but it is not advisable to aim for 100% utilization. Low resource utilization is also not good. However, there might be cases where low resource utilization cannot be improved. For example, there might be a short duration task in the business process for which a very specialized engineer is needed. If such an engineer does not want or is not capable of doing any other tasks then low resource utilization cannot be avoided. Fortunately, usually this is not an issue and low resource utilization can be fixed by either reducing the amount of resources or by assigning additional tasks.

1.1. Problem Statement

In this thesis I will attempt to generate an automated business process optimizer. The optimizer will try to optimize execution time, cost or combination of thereof when given certain resource constraints, expressed in terms of minimum and maximum size for each resource pool.

1.2. Case Study

Throughout this thesis, I will be using an insurance call centre case study. It is about an insurance company that has two call centres and a back-office for dealing with insurance claims. Each week an average of 9’000 calls are received in each call centre. There is also a busy period during the storm season where an average of 20’000 calls per week are received in each call centre. There are two types of workers: call centre agents and claim handlers. Call centre agents answer a call and deal with the same case until it gets declined or forwarded to the back office. Claim handlers are the ones who do the post-call

processing in the back-office. There is only one back office so claims from both call centres end up in the same back office. Currently there are 90 call centre agents working in each call centre and 150 claim handlers working in the back office. A call centre agent earns 2000 € per month (12.5€/h) and a claim handler earns 2500 € per month (15.5€/h).

Figure 1 shows the business process in EPC notation. Average task durations and gateway split probabilities are also shown.

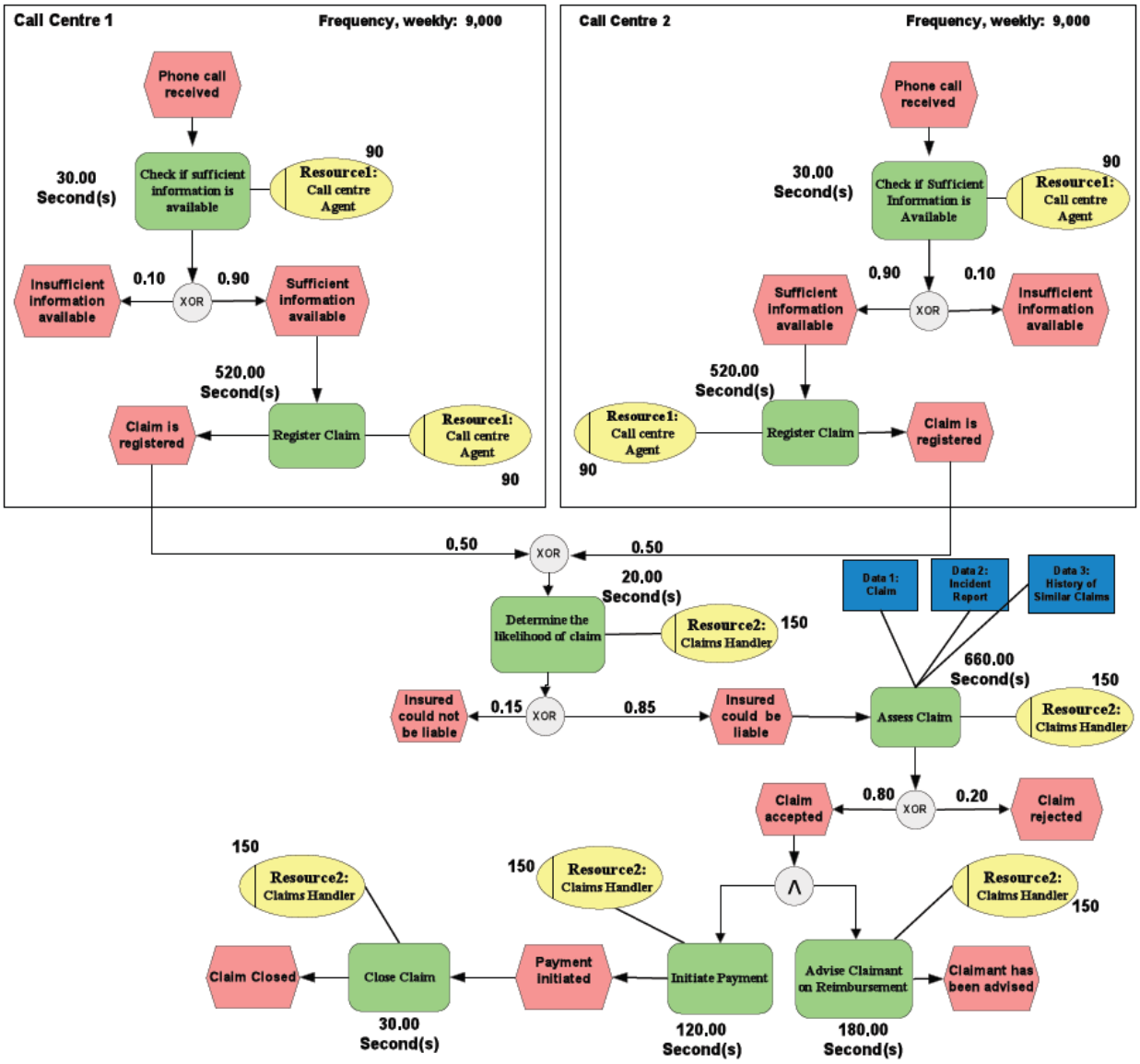

Figure 1: Insurance Claims Case Study

A detailed description of the case study can be found in [3].

1.3. Overview

The structure of the thesis is as follows. Chapter 2 presents related research and existing solutions to the problem introduced above. Chapter 3 presents the proposed resource optimization solution and the challenges faced and design choices made during the development of this solution. Chapter 4 discusses an evaluation of the solution using the case study introduced above. Chapter 5 concludes and outlines directions for future work.

2. Related Work

There has been a significant amount of research done in the area of business process optimization, but their main subject has been slightly different to mine. Unlike most other business process optimization papers I will attempt to optimize the process by purely manipulating resource allocation. In this respect, the focus of my work differs from the work reported in [1], where the aim is to suggest ways of optimizing the process by relocating tasks.

[4] addresses a resource allocation problem and in the problem formulation resources are grouped into roles, each role is capable of executing multiple tasks and a task can be performed by multiple roles. This scenario is very similar to the multi-resource allocation problem that I will address later in the thesis. The main difference being that in [4] the number of resources is not limited but in my approach the maximum number of resources can vary and the purpose is to find the optimal number of resources per role. Another way of explaining this difference is that [4] aims at optimizing resource utilization while I optimize total resource cost.

The literature on business process redesign abounds with examples of how simulation can be used to assess redesigned processes. Greasley [5, 6, 7] gives examples how a simulation tool, namely ARENA [8] can be used to evaluate a redesigned process. This paper provides a good demonstration of how simulation can be used to assess new business processes before implementing the changes in reality. Anyhow, these approaches do not address the problem of exploring large number of possible resource allocations in order to identify the one that strikes the best trade-off between multiple metrics such as cycle time and resource cost.

[9] gives an example where multiple deadline escalation strategies were applied to the same insurance claims case study that I will be trying to optimize. For example, in this paper the authors attempt to apply alternative paths to the process that have higher costs but lower execution times. Furthermore, early escalation is attempted which introduces an additional task where negotiations are held with the client when the process instance is being identified as being behind schedule. [9] is a good example where an automated resource optimizer could have been useful as in the paper only a few resource allocation configurations were used and therefore the true value of optimizing on resource allocation was not identified. [3] also covers various escalation strategies but the attention is on

deadline-based escalation. The paper focuses on changes that should be done when meeting the deadline is not realistic. Various escalation strategies like resource redeployment, data degradation and alternative path selection are covered and analysed through case studies.

Numerous attempts have been done to apply generic algorithms to process optimization. In [10] a genetic algorithm coupled with coloured petri nets was successfully used to minimize production and reconfiguration costs in a manufacturing system. In [11] a genetic resource optimization algorithm was successfully applied to two different case studies. The results in this paper are of additional interest to me as it also contains results for the same insurance claims study that I will use for evaluation in this thesis. Furthermore, like me, they focus on optimizing on resource allocation rather than process redesign or other escalation strategies. The case studies were used to prove that applying a genetic algorithm can create a significantly more efficient process compared to the original resource allocations.

Although genetic algorithms have been proven to find optimal resource allocations there are a few drawbacks to that approach. First, the genetic algorithm takes a lot time to run: around 20 hours in case of the [11]. Second, the algorithm is only capable of optimizing on one dimension. In the given scenario cycle time divided by cost was used. Due to these restrictions it is rather difficult to use a genetic algorithm approach in the real world. If a business analyst is interested in process redesign as well as resource optimization then he or she will not have enough time available to run multiple simulations. Furthermore, I believe that an important feature is the ability to see the trade-off between cost and cycle time. The only realistic way to express such relations is on a two-dimensional graph using cost and time as the dimensions. Figure 2 gives an example of how a *Pareto frontier* can be identified on a two-dimensional space.

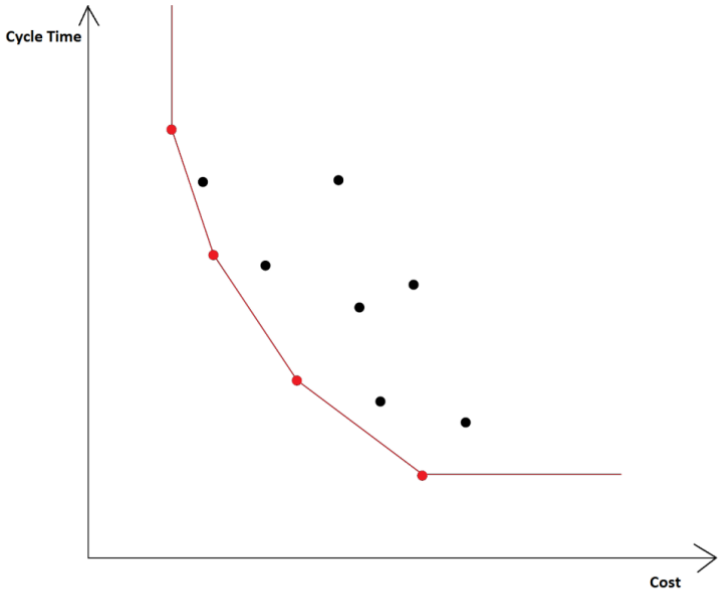

Figure 2: Pareto Frontier

A *Pareto frontier* [Figure 2] is a line where each point on its path is *Pareto efficient*. If a point on a 2-dimensional graph is located so that there is no way of improving one dimension without sacrificing the other then the point is regarded *Pareto efficient*. More information regarding *Pareto efficiency* can be found at [12]. Finding such points is mathematically expensive and usually requires calculating all input combinations. In case of business processes this would be manageable if one has a simple business process with one or two resource pools and a few tasks. On the other hand, in case of the insurance claims case study the problem becomes computationally impossible. In this thesis I will attempt to create an optimizer that is capable of finding a near-optimal *Pareto frontier* within reasonable time limits even for complex problems. Namely the objective is to identify a *Pareto frontier* for the insurance claims study using cost and cycle time as dimensions.

3. Business Process Optimizer

To reiterate the purpose of this thesis is to create a resource allocation optimizer for business processes. Given a range of permitted resource counts for each resource pool (i.e. the maximum and minimum size of each resource pool) the optimizer would be capable of optimizing the allocation of resources with respect to a given set of key performance indicators (KPIs). For example, the optimizer would attempt to find the optimal resource allocation for the lowest cost and/or lowest cycle time. There would be no limitations to the amount of dimensions that the implementation can optimize on. For example, an additional optimization dimension could be cycle time multiplied by cost. Another important consideration when implementing this optimizer was the fact that its execution time would be in the scope of a couple of minutes. With these requirements in mind I decided to tackle this optimization problem using a heuristic optimization technique called hill-climbing.

3.1. Hill-Climbing

Hill-climbing is an incremental algorithm that first finds a single solution to the problem and then tries to improve the solution by making small changes to the current solution. A useful feature of hill-climbing is that it is an anytime algorithm meaning that it does not have to end in order to have a result. As a consequence, each result obtained by the algorithm is a valid output and can be used to analyse the algorithm's behaviour and efficiency. Furthermore, hill-climbing is usually capable of finding a better solution than other optimization algorithms if the amount of time available for execution is limited. On the other hand, hill-climbing comes with a downside: namely it is good at finding the local optimum but there is no way to guarantee that the end result is a global optimum.

Other alternative meta-heuristics include Tabu Search, Genetic Algorithms (GA), Simulated Annealing and Ants Colony optimization. These algorithms explore a larger subset of the search space and may find more optimal solutions, at the expense of higher execution times. Since execution time is a major bottleneck in the context of business process resource optimization, I decided to focus on hill-climbing as an initial approach to this problem.

The high-level hill-climbing algorithm is described below:

```
curBest = result for a given/random set of inputs
improved = false
```

```
do
        improved = false
        for (all neighbouring input sets)
                curResult = result for current neighbour
                if (curResult > curBest)
                        curBest = curResult
                        improved = true
while (improved)
```

Code 1: Hill-Climbing Algorithm

For example, the hill-climbing algorithm can be used to solve the travelling salesman problem. In the traveling salesman problem neighbouring input sets are defined by swapping the order of one certain city with all other cities. The algorithm would finish when it identifies that moving one certain city to a new order location gives no improvement in the total distance travelled by the salesman.

Unlike my resource optimization problem the traveling salesman problem is in a certain way of a simpler nature as it focuses on optimizing only on one dimension: total distance travelled.

In order to apply hill-climbing to the resource-allocation problem the following questions need to be considered:

- How to assess the quality of a given resource allocation?
- How to obtain an initial resource allocation (i.e. a starting point for hill-climbing)?
- How to compute the neighbours of a given resource allocation?
- How to simultaneously optimize along both resource cost and cycle time?

Below I will discuss how I tackled each of these questions.

3.1.1. Simulation Engine

My hill-climbing resource optimizer needed a business process simulator in order to run a simulation for each possible resource allocation explored during the search. This simulation provides the performance statistics needed to assess the quality of the allocation. Since it is possible that many allocations need to be explored, it is crucial that each simulation is done as fast as possible. For this purpose, I make use of a fast BPMN process simulator developed in an earlier Master's thesis: [2]. In the examples given in this thesis, the simulator outperforms IBM WebSphere Business Modeler by more than ten times. Another alternative could have been to use CPN Tools [13] but I believe that from a user perspective BPMN is more intuitive than coloured petri nets.

3.1.2. Starting Point

In order to start hill-climbing one needs a base allocation which is used to initiate the climbing process. I have identified the following possible starting points:

1. Resource allocation defined by the user. If the business analyst is running the optimizer for process re-design purposes then these allocations could be the as-is resource allocations. For example, in case of the insurance claims case study the optimization could start with 90 people at both call centres and 150 claim handlers.
2. Start at a random point. This means that the optimizer will pick random resource allocations in the allowed resource allocation ranges. For example, if the user has defined that there can be 150 to 250 claim handlers then the program will pick a random number inside that range for the starting number of claim handlers.
3. Minimum allowed values. If a minimum of 150 claim handlers and 70 call centre agents are allowed then the starting point would be these values.
4. Maximum allowed values. The exact opposite of minimum allowed values
5. Start at multiple points. As hill-climbing algorithms are prone to finding the local optimum then it might make sense to have multiple starting points to increase the probability of finding the global optimum. For example, one could start an optimization process from four random points in the allowed ranges.

3.1.3. Identifying Neighbouring Allocations

If one had only one resource pool to handle then finding neighbours would be straightforward as the neighbours of 5 are 4 and 6. However, business processes usually involve multiple resources. This creates a situation where the amount of direct neighbours grows exponentially with the number of resources:

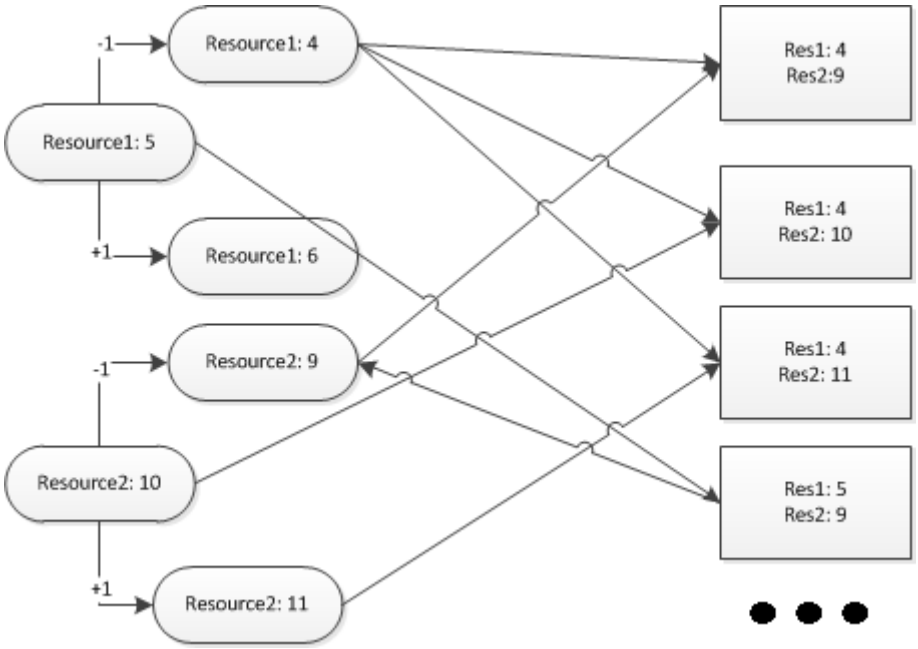

Figure 3: Neighbour Identification

So, in case of two resource pools eight possible neighbours are possible: 3 possibilities to the power of 2 resources and then the starting point must be subtracted. The table below shows how quickly the amount of potential neighbours rises with the number of different resources pools:

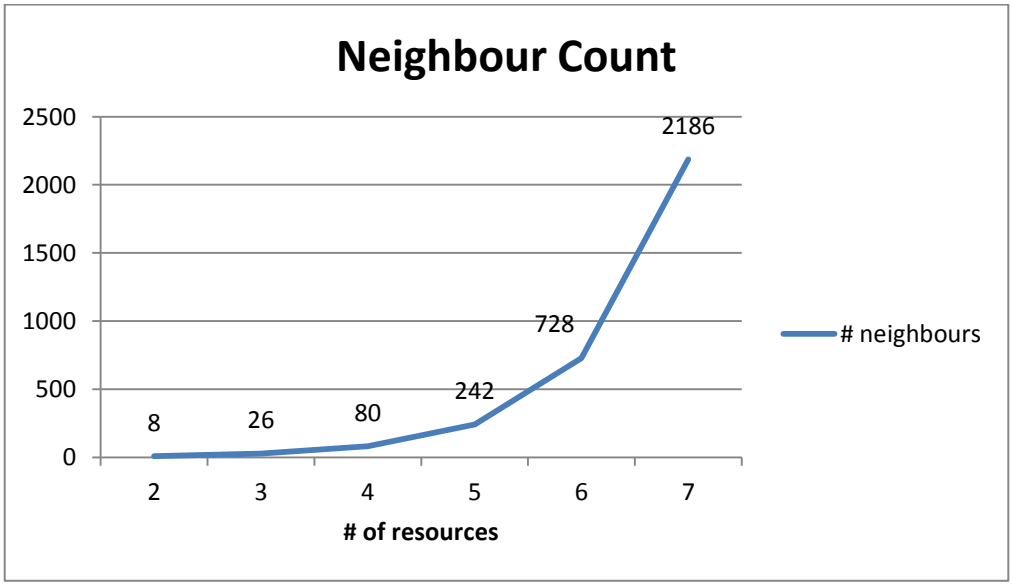

Figure 4: Neighbour Counts

The table above shows a significant limitation when using hill-climbing for the purposes of business process optimization. If one has a business process with 6 resources then even with a simulator that can run a single simulation in one second it would take around 12 minutes to calculate the KPIs for all the neighbours of one given resource allocation. In a

hill-climbing context this is an unacceptable result and it would take days to optimize one process with so many resources pools. Overall, I believe that hill-climbing can be applied to processes with up to 4 types of resource pools without sacrificing too much speed.

3.1.4. Optimizing on Multiple Dimensions Concurrently

An additional consideration was that the optimizer should be capable of optimizing on multiple dimensions. One of the initial solutions was that for each dimension a different optimization process was started and in the end the results of all the different dimension optimization could be combined to investigate the existence of a *Pareto frontier*. However, the optimizer was implemented using a more efficient algorithm that climbs on both dimensions concurrently. This is achieved by evaluating the neighbour resource allocations of the current resource allocation on multiple dimensions. This creates a situation where the hill-climbing can spread from one location into multiple new resource allocations.

An example scenario is described in the diagram below:

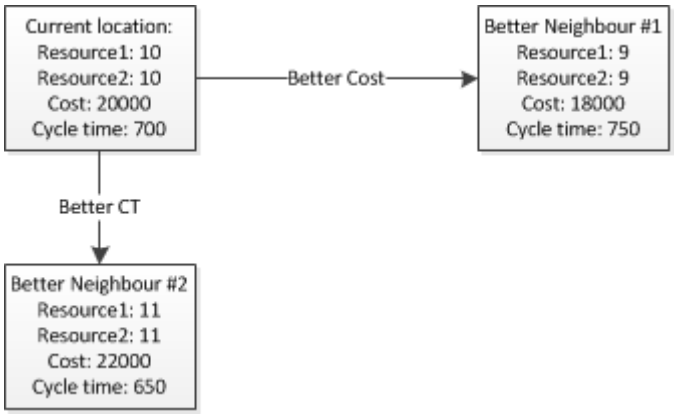

Figure 5: Optimizing on Cost and Cycle Time

Figure 5 demonstrates what happens when hill-climbing happens concurrently on two dimensions. In the first phase neighbours of 10-10 needed to be analysed. However, in the next iteration the optimization gets split up into two branches. This means that double the amount of neighbours have to be analysed. The result of the second iteration of the hill-climbing is shown below:

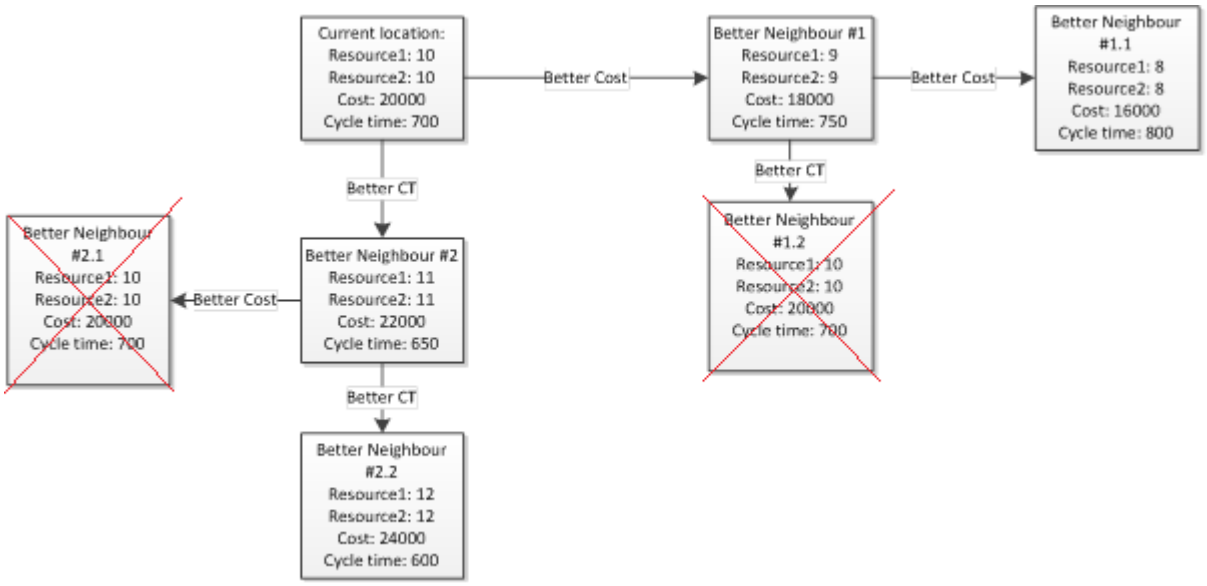

Figure 6: Iteration #2 of Optimization

This diagram demonstrates one of the problems that occurs while hill-climbing on multiple dimensions. When analysing "better neighbours" the optimizer is prone to go back to the starting position as it offers either better cycle time or cost depending on why the new neighbour was visited. To solve this back-looping issue I introduced the concept of blacklisting. This means that after a certain resource allocation has been simulated it will not be considered as a potential neighbour for any consequent resource allocation. Below is a diagram showing optimization phase 2 with the blacklist enabled.

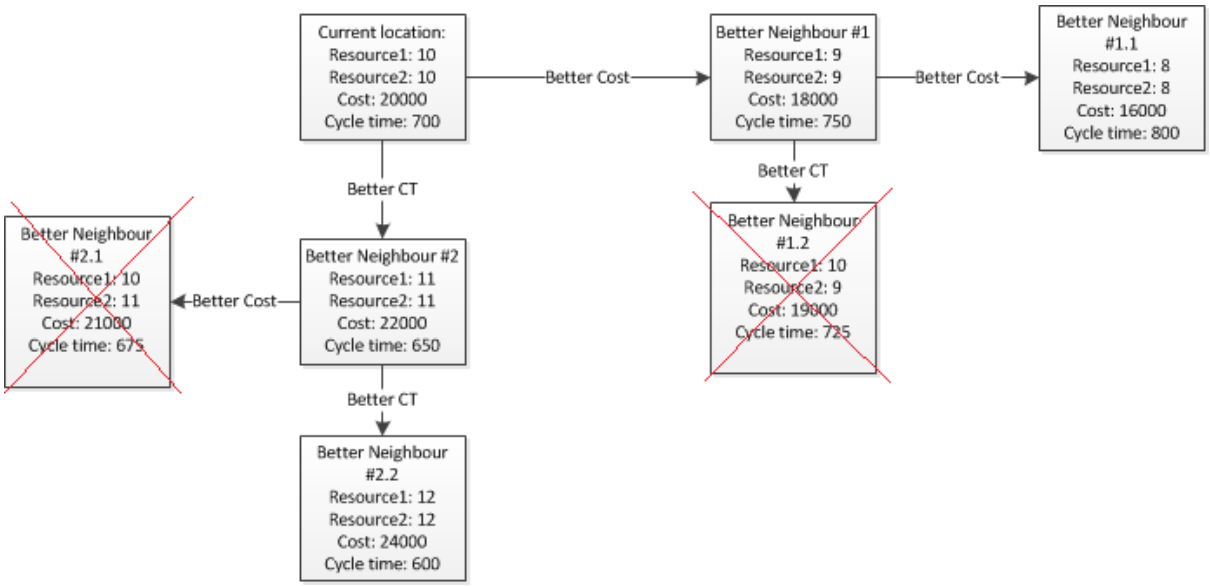

Figure 7: Optimization with Blacklist Enabled

As one can see then in this scenario the optimizer is still finding better cycle time and better cost as 10-9 and 10-11 are valid neighbours that have also not yet been blacklisted.

However, the optimizer already knows a resource allocation (10-10) that has a better cycle time than 10-9 and therefore 10-9 has no real value. Furthermore, allowing the optimizer to branch to 10-9 and 10-11 introduces two additional resource allocations that have to be analysed in the next iteration and each new resource allocation makes the optimizer slower. Below is an example how in the third iteration the branches from #1.2 are calculated but actually both better cycle time and lower cost exist in other branches.

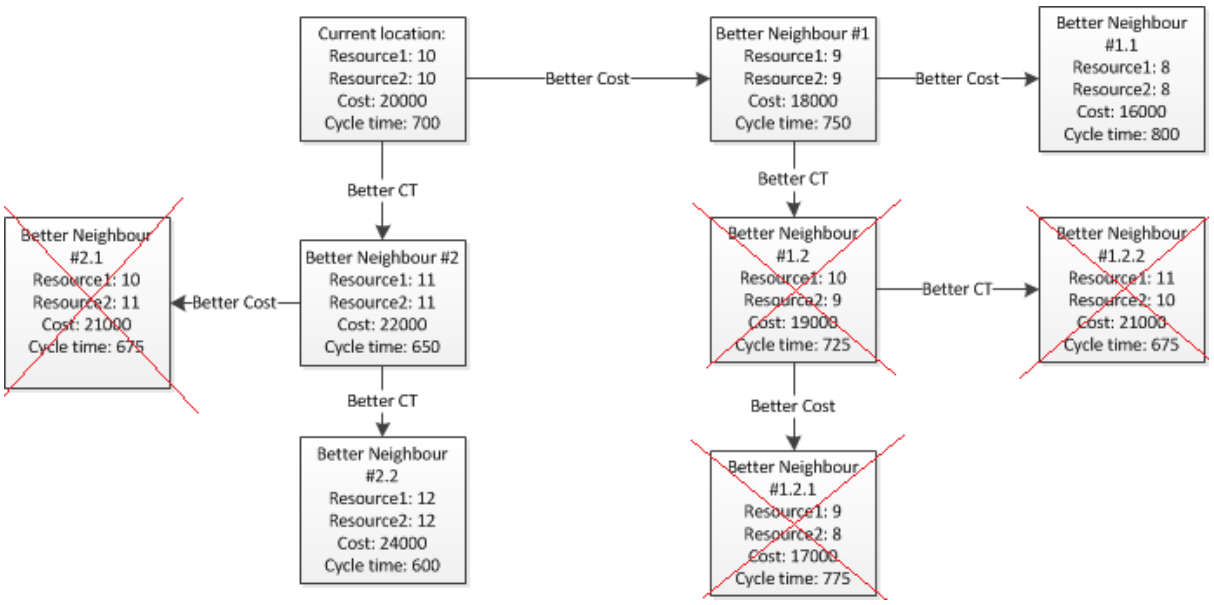

Figure 8: Optimization with Global Performance Measures

Figure 8 demonstrates that #1.2.1 and #1.2.2 are being identified as being improvements on both cycle time and cost but in reality #1.1 and #2.2 outperform both of them in their respective KPIs. Furthermore, they were discovered an iteration earlier. I do see that there might be some cases where such wondering might be useful in the search space, especially in case of a very high number of resources, but for our purposes such behaviour shout not be allowed. Performance is an important feature for my business process optimizer and workload that has doubtable value should be avoided. For this purpose I introduced the notion of a *Global Best Result*. This means that in order for a resource allocation to be regarded as an improvement it has to be the best result on a global optimization scale. For instance, #1.2 would get rejected in the previous example as a better cycle time was already discovered in #2 in the previous iteration.

3.2. Speeding-Up the Hill-Climbing via Configurable Climbing Intervals

Another feature of the business process optimizer that I managed to implement is the ability to modify the hill-climbing interval. This means that before executing the optimization the user can specify how close to the current resource allocation the optimizer should look for neighbours. If the user specified the climbing interval to be 5 then the neighbours of 10 would be 5 and 15 instead of 9 and 11. This feature is not useful when one is dealing with small resource counts but when optimizing a process where resource allocations are in the hundreds it usually does not make sense to optimize with a step of 1. Increasing the climbing interval has another useful side effect that makes the selection of the starting resource allocation less important. If the business analyst/user has problems estimating what the optimal resource allocation might be then he/she can just set the climbing interval to be larger. This avoids a situation where the analyst chooses an unsuitable starting point for the optimization and the simulation runs for hours in order to reach resource counts very far from the starting point.

Regarding climbing intervals I also implemented an additional feature that allows specifying a starting and ending interval. This means that one can set the starting interval to 50 and the ending interval to 1. What this means is that the optimizer will start with analysing neighbours 50 resource counts above or below the starting point. However, the optimizer does not terminate when it discovers that no improvement can be made in the +-50 range. Instead, it restarts the optimization at the latest improved resource allocation with a climbing interval of 25. Such behaviour continues until the optimizer has found no improvements with the end interval, this case 1. This is a useful feature when the user is interested in finding the best performing resource allocation for a specific KPI. The very low cost or very low cycle time is reached faster but the end result will still be rather accurate as the optimizer closes down on the most optimal resource allocation by using smaller and smaller climbing intervals. The declining climbing interval approach is however not very useful when the user is interested in finding the *Pareto frontier*. On a *Pareto frontier* the intermediate points are also relevant but the large climbing interval in the beginning of the optimization skips most of them.

Below are examples of one-dimensional hill-climbing using a standard climbing interval of 5 and a declining interval starting at 50 and ending at 1:

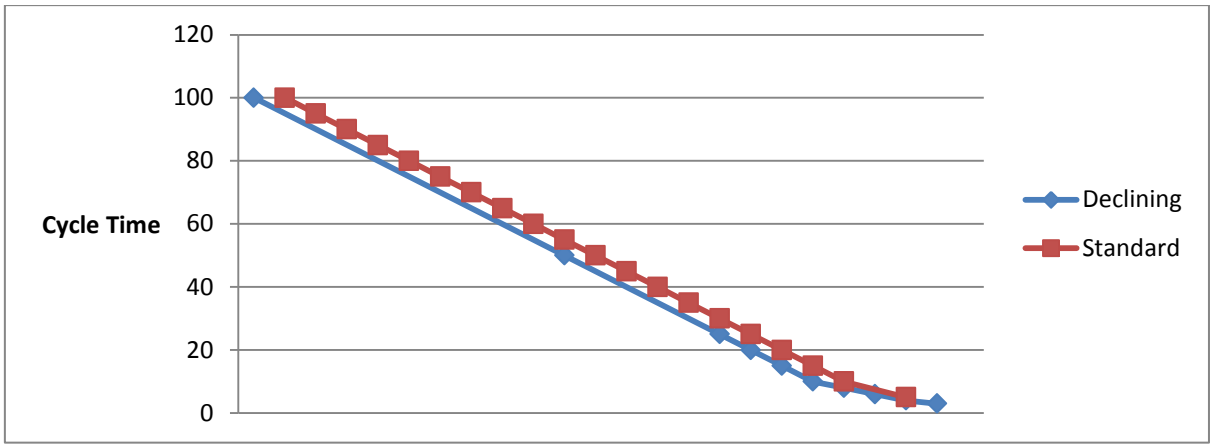

Figure 9: Small vs. Large Climbing Intervals

3.3. Handling Variance in Simulation Outputs

One of the most challenging problems that I faced when implementing the optimizer was the variance of simulation results. Each simulation has a number of attributes that play a huge role in the final outcome. The three causes of variance are gateway split probabilities, input arrival rate distribution and task duration distribution. These three create a situation where running a business process simulation with exactly the same resource counts will produce different results.

Figure 10 shows the average cycle times for twenty insurance claims case study simulations that were ran with the following parameters: 90 Sydney call centre agents, 90 Melbourne call centre agents, 180 claim handlers and 18000 calls per week.

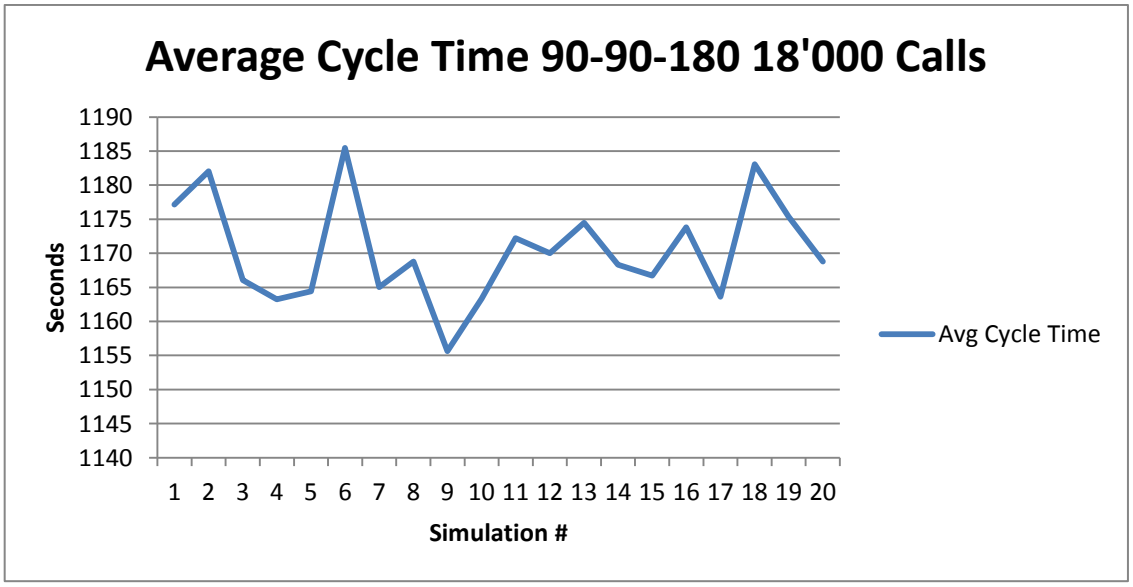

Figure 10: Cycle Time Variance 90-90-180

The mean of these simulations was 1170.4 seconds with a standard deviation of 7.4. This means that the 95% confidence interval is [1167; 1173].

Such variance is reasonable and expected when a simulation is run with so many variable parameters. However, for hill-climbing purposes such variance creates a critical problem. With this optimizer I am interested in comparing results of different resource allocations to each other but due to variance it is possible that the variance caused by the simulation parameters outweigh the improvement caused by adding additional resources. Below is an example of the same insurance claims case study run with 91-91-181 resources.

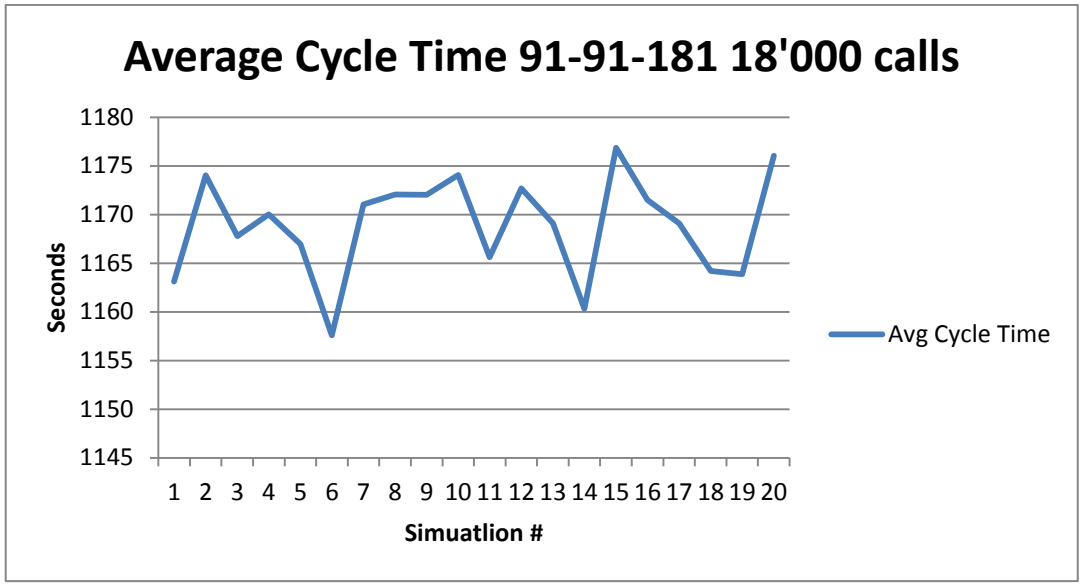

Figure 11: Cycle Time Variance 91-91-181

This time the mean is 1168.9 seconds with a standard deviation of 5 seconds. This adds up to a 95% confidence interval [1166.7; 1171.1].

As one can see then it is very possible that the 90-90-180 resource allocation outperforms the 91-91-181 in cycle time. In this case the optimizer would not identify that 91-91-181 is a better resource allocation regarding cycle time and would terminate. Although it might happen that 90-91-181 still outperforms 90-90-180 and therefore the optimizer moves to that location instead. However, this is unwanted behaviour from the perspective of efficiently finding the lowest cycle time resource allocation.

Furthermore, simulation result variance is even higher when the simulation is run in an unstable state. This includes the situations where there are very few resources allocated and as a result waiting times become much larger than working times. Below is an example

where the optimizer is run under the storm season scenario with 40’000 calls per week with two neighbour resource allocations.

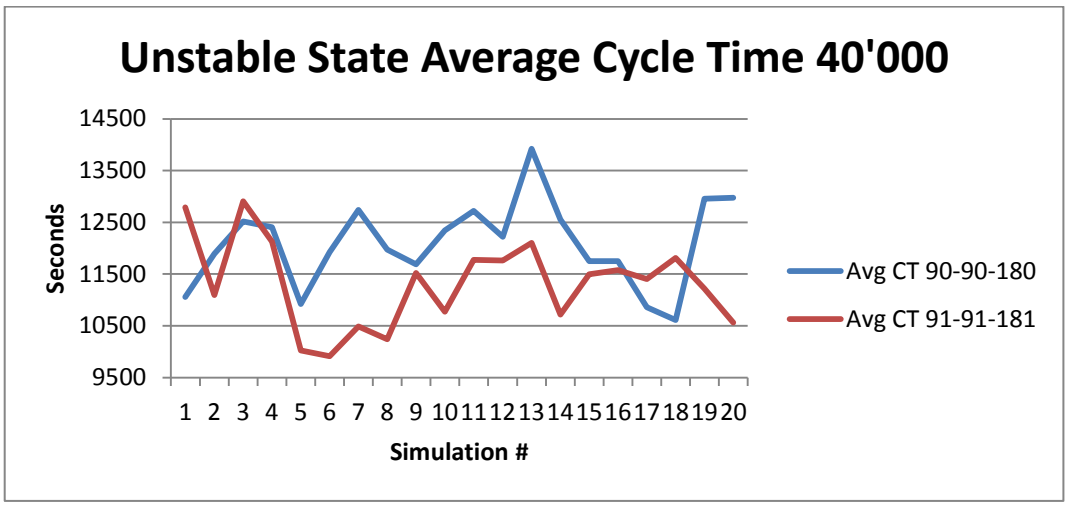

Figure 12: Cycle Time 90-90-180 vs. 91-91-181

The chart shows that in 4/20 cases 90-90-180 achieves a lower cycle time and might force the optimizer to terminate before reaching an optimal solution. Moreover, the standard deviation for 90-90-180 was 803 seconds from a mean of 12089 seconds. In the 18’000 calls per week scenario standard deviation was around 0.6% of the mean compared to 6.6% of the busier week. This shows that the probability of having a false termination of the hill-climbing process is more likely when the business process is simulated with insufficient resources. One can imagine a scenario where the user accidentally sets the starting resource allocations of the simulator to be in the insufficient range. In such a case the likelihood of the optimizer succeeding in finding an optimal cycle time solution is very low due to the high chance of false termination at each step of the hill-climbing process.

In order to compensate for the variance in simulations the following solutions were considered and attempted:

1. Using a larger climbing interval
2. Changing task durations and arrival intervals into fixed durations
3. Using the mean of multiple simulations
4. Implementing confidence intervals
5. Using *Simulation Replay*

Below I will expose each solution in turn.

3.3.1. Using a Larger Hill-Climbing Interval

The main cause of resource allocation comparisons returning invalid results is when new resource allocations do improve the results but not enough to overcome the variance. One easy solution to avoid such situations is to use a higher climbing interval. Below are examples of using a climbing interval of 5 and 10 on the stormy season scenario.

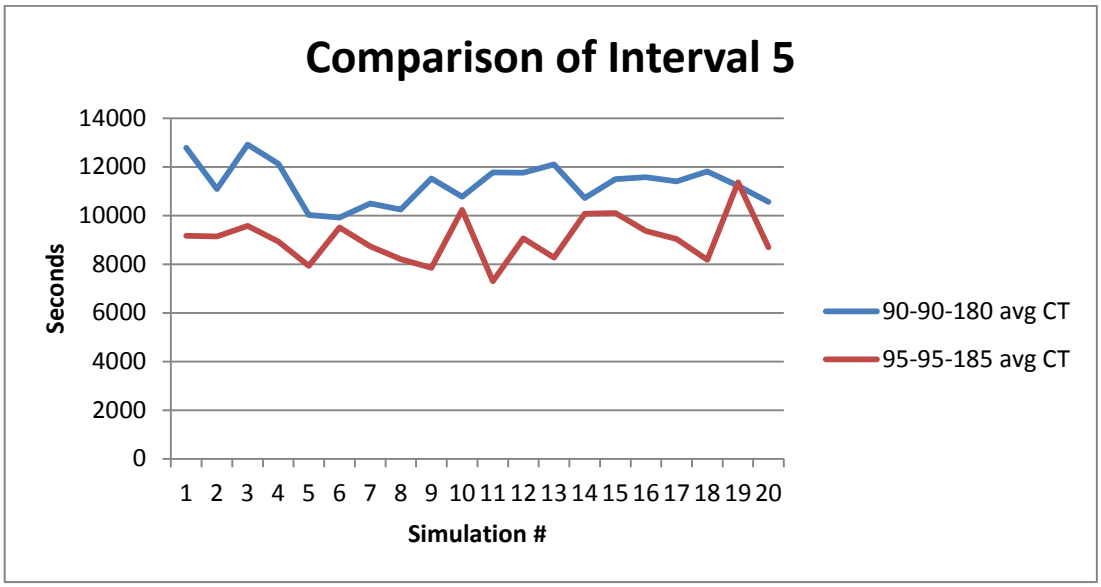

Figure 13: Variance with Interval 5

Although the mean of 90-90-180 is 11314.8 seconds and the mean for 95-95-185 is 9036.6 seconds then simulation number 19 shows that incorrect classifications can still happen.

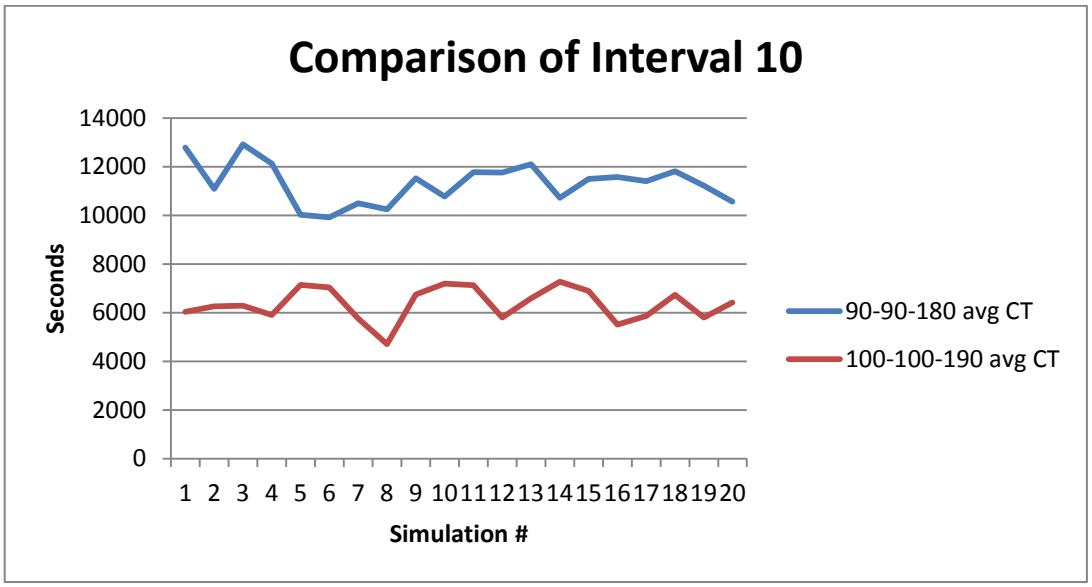

Figure 14: Variance with Interval 10 #1

Figure 14 shows that for the purposes of climbing from 90-90-180 with an interval of 10 it is very likely that 100-100-190 will be identified to have a lower cycle time.

However, this does not prove that an interval of 10 is the correct one to choose for optimizing the insurance claims case study. Below is an example where an interval of 10 still causes invalid classifications.

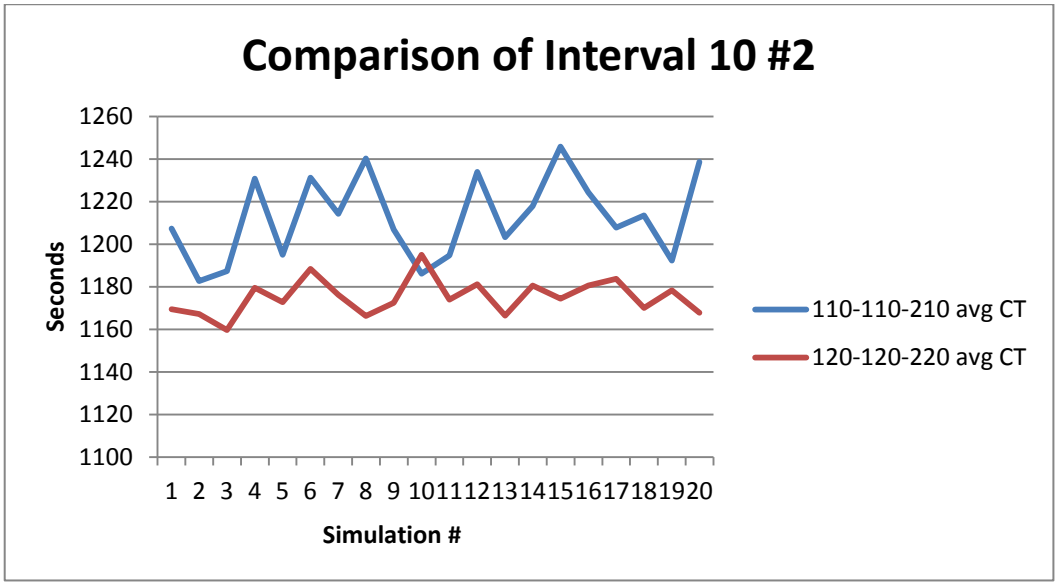

Figure 15: Variance with Interval 10 #2

Figure 15 shows that after getting close to the optimal cycle time an ever larger interval is needed to compensate for the variance in simulation results.

Another downside of having a configurable climbing interval for the purposes of beating variance is that the user has to be capable of reasoning about what kind of interval he/she should enter. The above examples even show that for each situation a different climbing interval is needed. Furthermore, the downside of using larger intervals is that the optimization results will be very general. 10 additional workers might mean an additional million euros added to the annual wage bill and this is generally not acceptable. Due to these reasons it is not advisable to use higher climbing intervals for its variance-beating characteristics.

3.3.2. Changing Task Durations and Arrival Intervals into Fixed Durations

Another optimization that I attempted was to change the task durations and the arrival intervals to fixed durations. In the real world tasks usually take some normally distributed range around the mean and arrivals/calls of customers are never evenly distributed. However, for my optimization purposes I believed that the gain on better hill-climbing

quality would overcome the downside of the simulations not being truly realistic. Below are the results of using fixed durations and arrival intervals compared to the realistic simulation where durations are distributed around the mean. The standard 90-90-180 resource allocation scheme with 40'000 calls per week was used.

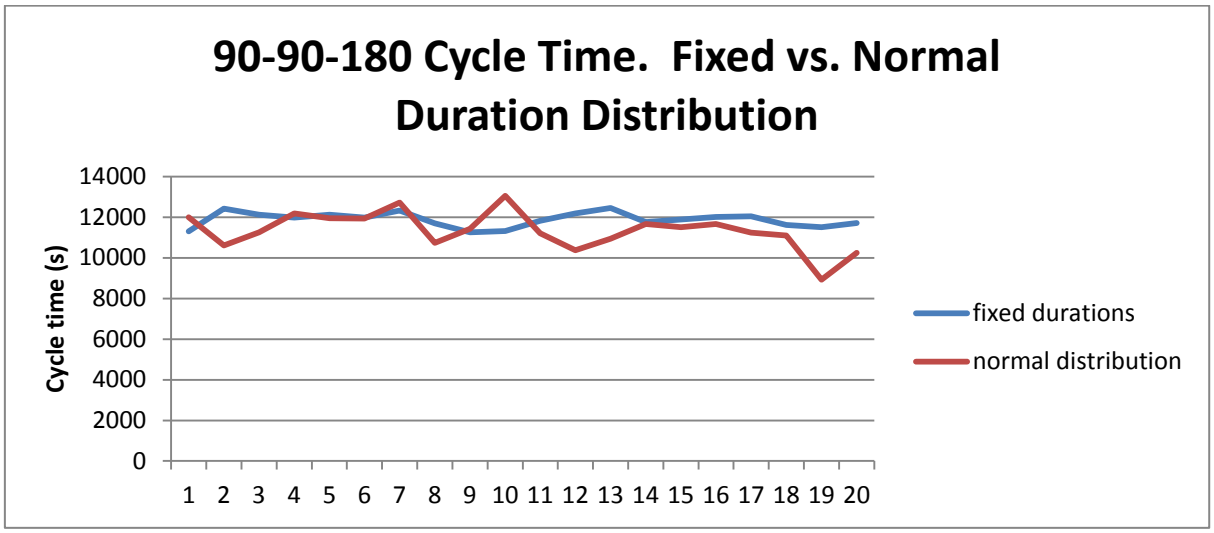

Figure 16: Fixed vs. Normal Duration Distribution Variance

The graph shows that fixed duration simulations have a significantly lower variance. However it is also important to note that the average cycle time for the two different simulation methods was very different and the confidence intervals did not actually overlap.

	Fixed Durations	Normal Duration Distribution
Mean CT	11875.28	11336.27
Standard deviation	347.7605024	897.1824985
95% interval	[11722.8;12027.7]	[10943;11729.5]

Table 1: Fixed vs. Normal Duration Distribution Variance

Although the fixed durations did lower the variance in the simulations this optimization technique was still not satisfactory as fixing the duration and inter-arrival times is not realistic, and simulating under this assumption leads to very different results compared to simulating under the more realistic assumption of variable durations and inter-arrival times.

3.3.3. Using the Mean of Multiple Simulations

The third optimization that I attempted was to run each resource allocation multiple times and use the mean of those simulations to compare it to other resource allocations. This however, is a very costly technique as the amount of simulations that have to be run will increase. For example, if one decided to simulate each resource allocation 3 times then the optimizer would become 3 times slower. To justify such a sacrifice I investigated the effect of using mean values for climbing. Below is a chart showing the variance in the mean of three simulations when run with the default 90-90-180 resource allocation.

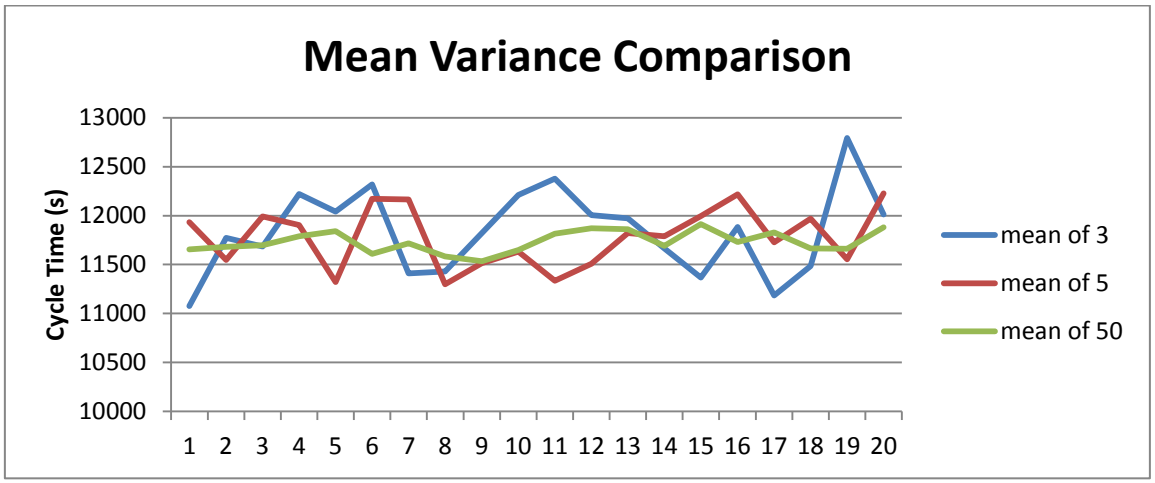

Figure 17: Mean Variance Comparison

As it becomes obvious from Figure 17 then even running simulations three times and using the mean of those will probably not work as the variance is still rather high. Below are the mean, standard deviation and the 95% confidence interval to show the observation. *Mean of 50* does demonstrate that this technique could be effective but as I am interested in a fast optimizer then calculating everything 50 times is not acceptable.

	Mean of 3	Mean of 5	Mean of 50
Mean	11836.61901	11781.91647	11733.75
Standard Deviation	423.8636335	298.0434651	106.4942
95% Interval	[11650.9;12022.38]	[11651.3;11912.5]	[11687;11780.4]
95% Start	11650.85599	11651.29556	11687.08
95% End	12022.38202	11912.53737	11780.42
CONF+-	185.7630144	130.6209076	46.67229
STDERROR	97.24098653	68.37586026	24.43145

Table 2: Mean Variance Comparison

In the end this technique was not the one used for evaluating the business process optimizer but it was the first approach that managed to improve the quality of hill-climbing.

3.3.4. Implementing Confidence Intervals

The fourth consideration regarding variance was an enhancement of the mean calculation approach. Here I attempted to join the mean calculation with confidence intervals. In this approach a resource allocation would be simulated in the following way:

1. Simulate each resource allocation x times
2. Calculate the mean and standard error of those runs
3. Use the mean and standard error to calculate the confidence interval for each allocation.
4. Compare the confidence intervals in order to decide if any of the neighbours of the original resource allocation improves the cycle time.

Comparison of confidence intervals would be done in such a way that if the intervals overlap then the comparison has instantly failed and one or the other would not be regarded as a better solution. Such interval comparison is a widely used approach in business process simulation. For instance, confidence intervals were used to evaluate a redesign of a road traffic accident reporting system in [6]. I was, however not very confident about the applicability of confidence intervals in the context of hill-climbing. I had already shown in the previous section that when using mean values over multiple simulations then the results still have significant variance.

After implementing the confidence interval approach it became obvious that this technique is indeed not applicable in the context of hill-climbing. I ran a sample scenario where each resource allocation was calculated 5 times.

	mean CT of 5 simulations	**95% confidence interval(CI)**	**90% CI**	**85% CI**	**MEAN**	**95%**	**90%**	**85%**
90-90-180	11833.64	[11295.35; 12371.94]	[11380.48; 12286.8]	[11482.1; 12185.18]				

90-91-180	11889.17	[11584.97; 12193.37]	[11633.08; 12145.26]	[11690.51; 12087.83]				
90-90-181	11674.19	[11040.84; 12307.54]	[11141.01; 12207.36]	[11260.57; 12087.8]				
90-91-181	11868.75	[10693.44; 13044.05]	[10879.33; 12858.16]	[11101.2; 12636.29]				
91-90-180	12035.08	[11683.32; 12386.84]	[11738.96; 12331.2]	[11805.36; 12264.8]				
91-91-180	12287.69	[11870.85; 12704.53]	[11936.78; 12638.6]	[12015.47; 12559.91]				
91-90-181	10899.05	[10426; 11372.1]	[10500.82; 11297.28]	[10590.12; 11207.98]	**choose**		does not collude, **choose**	does not collude, **choose**
91-91-181	10900.48	[10132.03; 11668.93]	[10253.57; 11547.39]	[10398.63; 11402.32]				does not collude

Table 3: Confidence Interval Analysis

The simulation result show that if a confidence interval of 90% was chosen then there is some hope that the optimizer will find a suitable neighbour and the hill-climbing can continue. When one would simulate with 85% confidence the possibilities for continuing are even higher. By continuing I mean that the optimizer finds a new resource allocation that has a lower cycle time than its own confidence interval.

However, when climbing with an interval of 1 I believe that the probability of the optimizer failing to find a non-overlapping confidence interval is too high, even with 85% confidence.

A viewpoint could be that if one used larger hill-climbing intervals then it might actually be possible to use the confidence intervals method to compare various resource allocations with each other. However, then one would be facing exactly the same problems as with just climbing with larger intervals. It is not realistic to expect the user to be capable of making an educated guess about the most suitable climbing interval. All things considered

the confidence interval method was also not selected as the comparison technique for the business process optimizer.

3.3.5. Using Simulation Replay

The fifth and final approach that was tried was to use the same simulation structure for all simulations. In other words, the simulation is run once for the initial resource allocation (priori to hill-climbing) and during this simulation the following is recorded:

- The exact creation time of each process instance
- The exact duration of each task execution
- The outgoing path chosen when a given decision node is visited as part of the execution of a process instance (each time the node is visited)

Subsequent simulations are performed not stochastically, but rather by *replaying* the data recorded during the first simulation. To give an example, if in the first simulation the first three process instances were started at 14:01, 14:03 and 14:09 then in the following simulations the first three process instances would be started at exactly the same time. The same principle would be applied to gateway selections and task durations. Such an optimization will create a situation where the same resource allocation will always create exactly the same simulation result as variance between two simulations has been taken out of the equation. Most importantly this is supposed to create a situation where different resource allocations are directly comparable. The only downside is the fact that the optimization will be based on one simulation flow but in reality there might be millions of different combinations of gateway selection, start times and task durations available. On the other hand, this is a much better generalization than fixing all task durations and setting the process start events to be evenly distributed. Below is a graph showing the expected behaviour regarding cycle time when using one simulation version for all resource allocations.

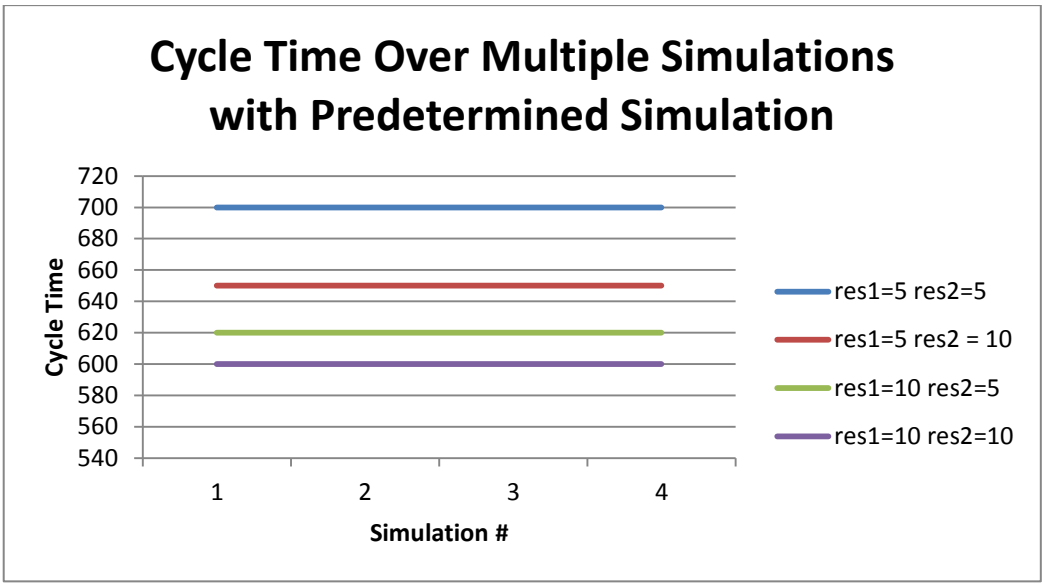

Figure 18: Cycle Time with *Simulation Replay*

The graph shows the general principle that I hope to achieve by using simulation replay. The results of using this technique can be found in the evaluation section of this thesis.

3.4. Extension to Multi-Resource Optimization

During the implementation process it occurred that another useful feature would be if the user could specify in the simulation parameters multiple resources that can do one certain task. An example is a shelving task in a library. As it is a task that requires no real qualification then probably any library employee could do the task. However, it might be the case that an additional person has to be hired because a lot of shelving has to be done. So it would be a really useful feature if the resource allocation optimizer could be run in a manner that it could be capable of analysing whether the current library employees are sufficient to complete all the shelving tasks. What I wanted to achieve is that the user can assign any amount of resources to one task. This creates a situation where one resource pool can be fully occupied and another can still do the task. There are however some tasks that need specialist attention and cannot be distributed to other workers. On the positive side multiple resource allocation is useful as it might be interesting to assign junior and senior staff to tasks. It would be the case that junior workers are paid less but they work less efficiently. This is another real-life problem where I believe my optimizer can be helpful in determining the correct ratio between senior and junior staff.

The only problem with BIMP was that it only supported assigning one resource to a single task. Fortunately this was solved by the fact that I had access to the source code and could make the required modifications in order to run my desired multi-resource simulations.

Furthermore, I implemented the new resource allocation in a manner that resources could be assigned tasks based on priorities. This is important as one generally prefers the person most capable of doing the task to be selected as the executor whenever possible. So in case of the library book shelving task then a person hired specifically to do this job should be preferred to a receptionist or any other worker. Only when the person with the highest priority for the task is already occupied will other resources be considered.

The high-level pseudo-code for multi-resource allocation is given below:

1) Check which resources can execute the task
2) In ascending order based on priority check if any resource is available to execute the task
3) If a resource is available then assign the task to him/her
4) If no resources are available put the task into the waiting queue for all the resources that can execute the task
5) If any of the resources comes available assign the task to that resource and remove the task from the waiting queue of the other resources

Such an implementation created a situation where priorities were not taken into account when a task was in the waiting queue. One future development could be that if a non-highest-priority resource was the first one to become available then it would only execute the task if none of his/her highest priority tasks were in the waiting queue.

Implementing the support for multi-resource allocation enabled me to address a lot of relevant research questions, some of which are analysed in the next section of this thesis.

4. Evaluation

In this section my optimizer will be evaluated on the basis of speed and quality.

For evaluation I will be using the insurance claims case study that was described in the introduction of the thesis.

Please note that all simulations were run on a standard desktop computer (Pentium i7-920, 8GB DDR3 ram). To speed up simulations eight threads were allowed to concurrently simulate various resource allocations.

4.1. Single Resource Allocation

4.1.1. Finding Optimal Fixed Duration Cost & Cycle Time

In this section of the evaluation, the cost dimension will represent the wages of all the assigned resources for one week (40h) and average cycle time is the average duration of a process instance.

Optimization 1*: Predefined Starting Point. Interval 1. 18'000 Process Instances*

The most general use case of the optimizer would be that the user models the as-is model in BPMN, annotates the model with the appropriate resource and duration distributions and then runs the optimizer with the as-is (default) resource allocation. The parameters for the first optimization will be the following:

1	Starting Point	DEFAULT(90-90-150)
2	Melbourne Agents (start-min-max)	90-50-150
3	Sydney Agents (start-min-max)	90-50-150
4	Claim Handlers (start-min-max)	150-100-250
5	Climbing Interval	1
6	Use Simulation Replay	FALSE
7	Call Count	18000

Table 4: Optimization #1 Parameters

The first parameter defines the location where the climbing starts from. The second, third and fourth parameter define the ranges in which the hill-climbing can occur. The fifth parameter specifies the climbing interval. The sixth parameter defines if the simulation should be run using the anti-variance technique introduced in the previous chapter where

the first simulation path is stored and reused for each succeeding simulation. The seventh parameter defines the number of calls that are received each week.

The results of running the simulation with the given parameters are given in Figure 19 and Table 5.

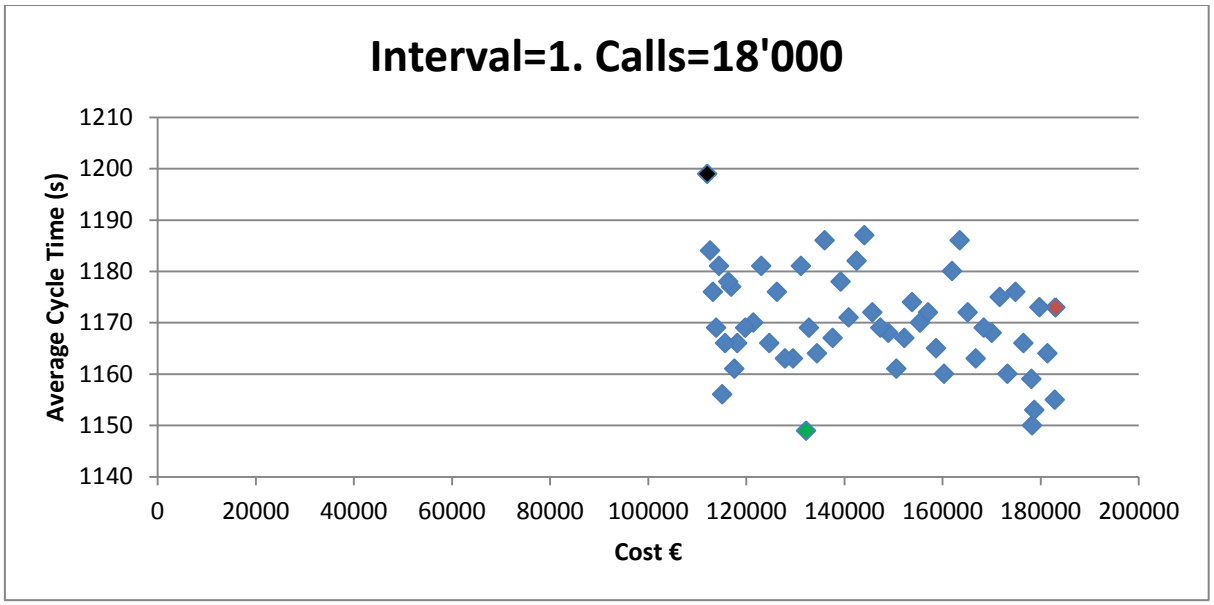

Figure 19: Optimization Results with Interval 1 and 18'000 calls

Lowest CT:	**1149 s**	**59-59-118**
Lowest Cost:	112000 €	50-50-100
Simulation Run Time:	195 s	

Table 5: Optimization #1 Results

These results demonstrate what happens if the optimizer is run with incorrect parameters.

First, the allowed ranges were incorrectly entered. As a result the hill-climbing algorithm never managed to climb to a resource allocation where there is a shortage of some resource. This meant that all the simulation results are very close to the optimal cycle time.

Second, the climbing interval was set to too low. This has two negative aspects. The main problem is that due to the fact that simulation results have variance in them it is likely that adding a few additional resources will not reduce the cycle time. In this 18'000 call scenario one single additional call centre agent in Sydney will probably not have a significant effect as the system is under-loaded. This means that the average cycle time is more dependent on the arrival rate of insurance claim calls and other variable attributes of a business process than the amount of resources assigned. The other side-effect of such variance also came evident in this optimization example: the optimizer only moved

towards better cost and not better cycle time. This is caused by the fact that there are already too many resources and additional resources will make not have much effect on the average cycle time. If I added 10 additional claim handlers then the cycle time will reduce noticeably but when only attempting to add one resource the result will not always be evident.

Optimization 2: *Predefined Starting Point. Interval 5. 18'000 Process Instances. Low Resource Ranges*

In order to address one of the shortcomings of the first simulation the interval was increased to 5 for this optimization. Also, the ranges of allowed resource allocations were reduced in order to reach higher average cycle times which were impossible in the first optimization. The results of the hill-climbing optimization are shown below.

1	**Starting Point**	**DEFAULT(90-90-150)**
2	Melbourne Agents (start-min-max)	90-20-100
3	Sydney Agents (start-min-max)	90-20-100
4	Claim Handlers (start-min-max)	150-50-150
5	Climbing Interval	5
6	Use Simulation Replay	FALSE
7	Call Count	18000

Table 6: Optimization #2 Parameters

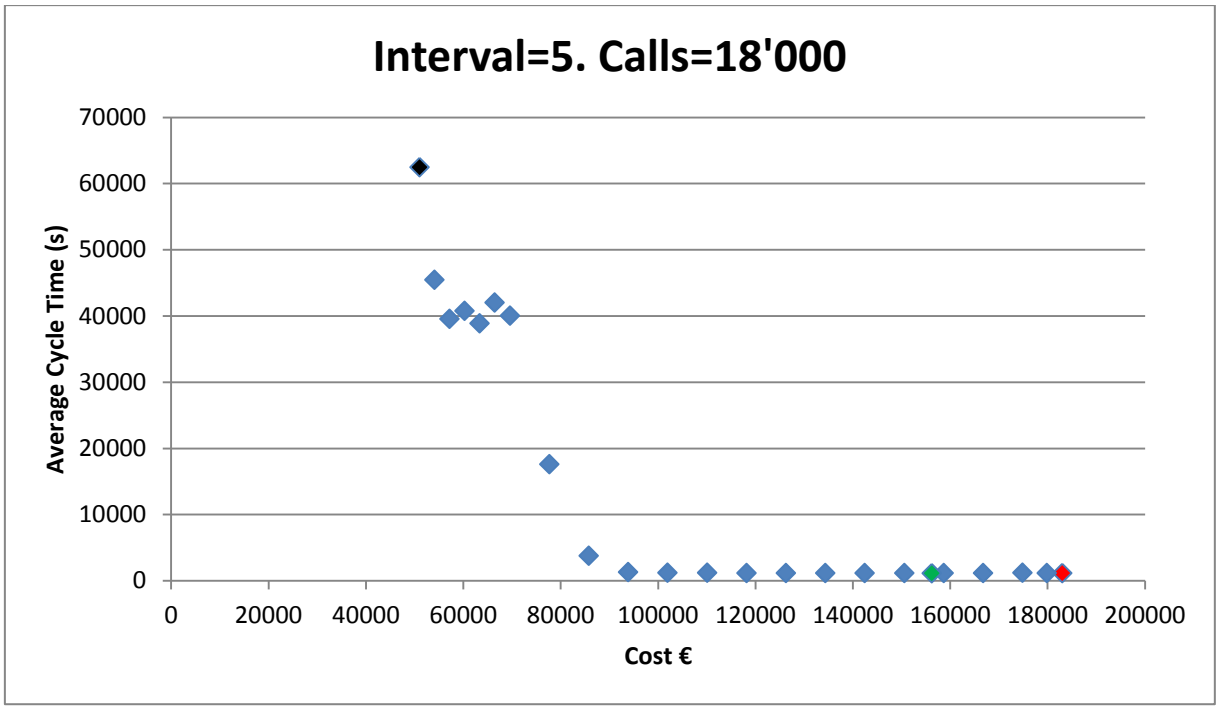

Figure 20: Optimization Results with Interval 5 and 18'000 calls

Lowest CT:	1153 s	70-75-135
Lowest Cost:	51000 €	20-20-50
Simulation Run Time:	74 s	

Table 7: Optimization #2 Results

Figure 20 is a fairly good representation of using a hill-climbing algorithm for optimizing a business process. Now it is possible to see that when the amount of resources are reduced then the average cycle time increases but the cost decreases.

On the downside the optimizer failed to move towards higher resource allocations as it was already achieving very low average cycle times in the starting position (red). Also variance caused the lowest cycle time to be achieved by a non-maximal resource allocation scheme (70-75-135).

The hill-climbing path of this optimization can be found in Appendix A.

Optimization 3: *Lower Starting Point. Interval 5. 18'000 Process Instances. Low Ranges*

As the starting point seems to play an important role in the outcome of the simulation I decided to experiment with various positions. Below are the results of running the previous optimization scenario with a 40-40-70 starting point.

1	Starting Point	DEFAULT(40-40-70)
2	Melbourne Agents (start-min-max)	40-20-100
3	Sydney Agents (start-min-max)	40-20-100
4	Claim Handlers (start-min-max)	70-50-150
5	Climbing Interval	5
6	Use Simulation Replay	FALSE
7	Call Count	18000

Table 8: Optimization #3 Parameters

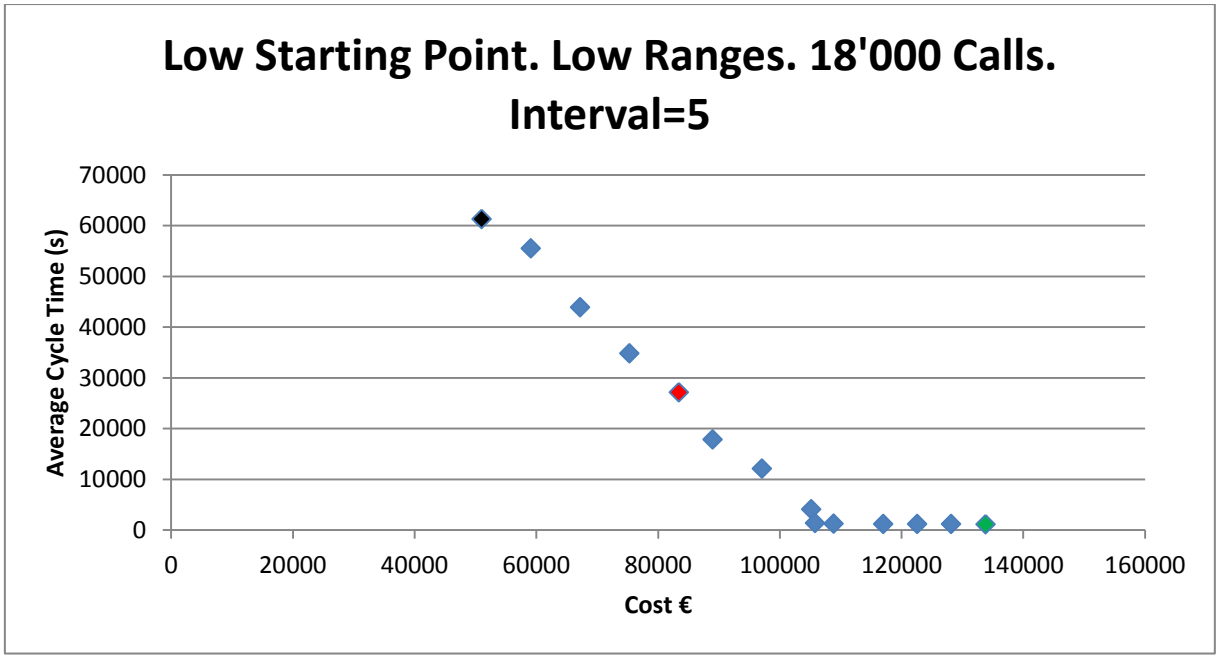

Figure 21: Optimization Results with Interval 5, 18'000 Calls and Low Search Ranges

Lowest CT:	1155 s	50-75-115
Lowest Cost:	51000 €	20-20-50
Simulation Run Time:	54 s	

Table 9: Optimization #3 Results

This is the first example where optimization was actually done in both dimensions: cost and cycle time. The optimizer climbed towards a better cycle time until it reached a point where the improvements became insignificant compared to the simulation variance.

Claim Handlers	Melbourne Call Agent	Sydney Agent	Call	Cost	Average CT
95	40	60		108900	1230
100	45	65		117000	1167

105	50	65	122600	1163
110	50	70	128200	1162
115	50	75	133800	1155

Table 10: Sample Hill-Climbing Steps for Optimization #3

In the partial hill-climbing result in Table 10 one can notice another useful feature of hill-climbing. Fairly low average cycle time (1230 s) is already reached with a resource allocation of 40-60-95. Choosing this allocation instead of the most optimal (50-75-115) is over 20% cheaper but only 7% slower.

Optimization 4*: Predefined Starting Point. Interval 5. 40'000 process instances*

To conclude a larger climbing interval was significantly more successful when optimizing the resource allocation for the 18'000 calls per week insurance scenario. As a result, the larger interval will be our selection for optimizing the storm season scenario where there are around 2.5 times more calls. Furthermore, one can expect the system to get overloaded under the increased load and therefore larger optimization ranges were re-allowed to allow the optimizer to reach near-perfect cycle times.

1	**Starting Point**	**DEFAULT(90-90-150)**
2	Melbourne Agents (start-min-max)	90-50-150
3	Sydney Agents (start-min-max)	90-50-150
4	Claim Handlers (start-min-max)	150-100-250
5	Climbing Interval	5
6	Use Simulation Replay	FALSE
7	Call Count	40000

Table 11: Optimization #4 Parameters

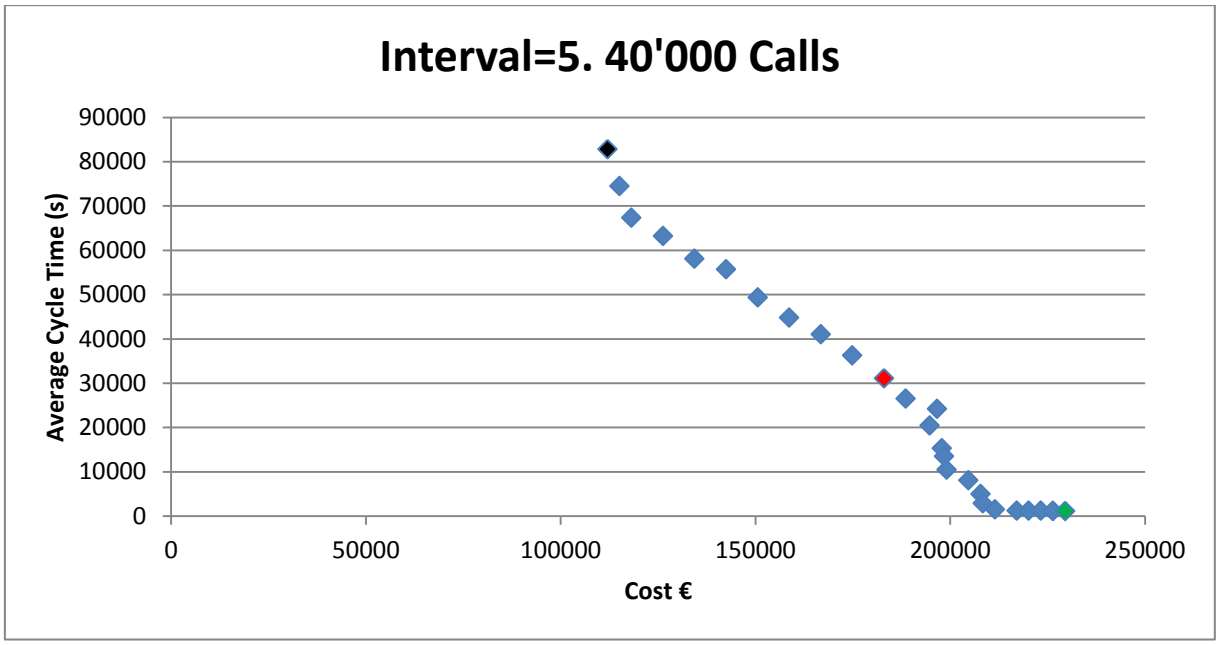

Figure 22: Optimization Results with Interval 5 and 40'000 Calls

Start CT:	**31126 s**	**90-90-150**
Start Cost:	183000 €	90-90-150
Lowest CT:	1161 s	85-95-220
Lowest Cost:	112000 €	50-50-100
Simulation Run Time:	162 s	

Table 12: Optimization #4 Results

Running the optimization with 40'000 calls per week creates a similar situation to the previous optimization where improvements on both dimensions are possible. The graph above shows how through hill-climbing the lowest cycle time (green) and lowest cost (black) were reached.

To conclude this section of the evaluation I can confirm that it is possible to determine the optimal cycle time for a business process. On the downside, such optimizations do give results but it is not proven that they are the best results. For example, in Optimization 4 it is not proven that 89-95-220 is the cheapest way to achieve an average cycle time of around 1160 seconds. Furthermore, using a static one week cost function is not very useful. In the insurance case study it makes sense until the total duration of the business process does not exceed 1 week. If the average cycle time of a process instance is 20'000 seconds then certainly all of the 18'000 or 40'000 insurance claims will not be handled in a week and will get delayed to subsequent weeks. As new calls will also be received in the upcoming weeks it will create an infinite queue and this is neither acceptable nor realistic.

4.1.2. Finding Optimal Process Duration Cost & Cycle Time

In the following sections of the evaluation I will use process duration cost and average cycle time as the optimization dimensions. This means that unlike the previous section the cost will also depend on cycle time. I am making the assumption that when all calls cannot be answered during normal working hours then all call centre agents and claim handlers have to work overtime to handle all claims. This will potentially create a situation where reducing resources will actually increase the cost of the process when a shortage of one type of resource will force others to work overtime.

Optimization 5: *Interval 5. 18'000 calls. Duration Cost*

In this section I will evaluate the new cost function using the same parameters as in optimization 3. The results of the optimization are below:

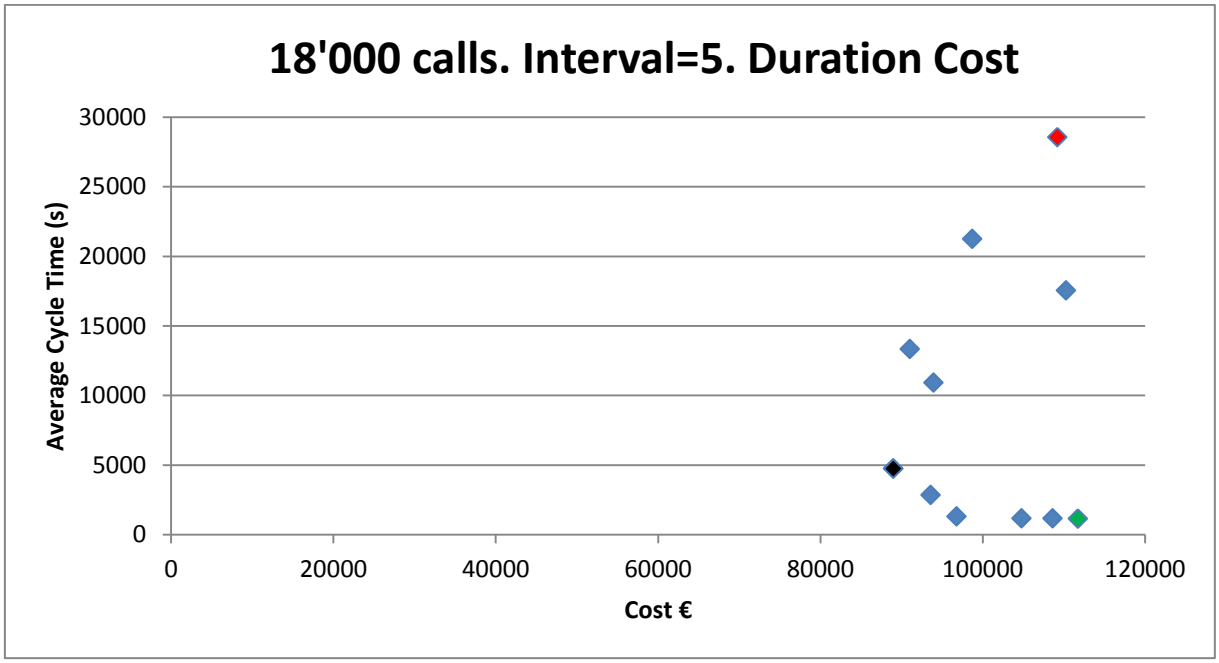

Figure 23: Optimization Results with Interval 5, 18'000 Calls & Duration Cost

Lowest CT:	**1162 s**	**40-40-110**
Lowest Cost:	88979 €	30-30-85
Simulation Run Time:	46 s	

Table 13: Optimization #5 Results

Using total process duration in the cost estimations creates a very different result. The duration cost approach sometimes creates a situation where optimizing on cost also comes with an improvement on cycle time. In the context of this optimization it means that the preselected start position 20-20-50 was a resource allocation that caused large delays and

therefore forced the total process duration cost to be very high. The most significant achievement of this optimization is that it managed to achieve a very low cycle time (1162 s) without increasing the cost compared to the start position.

Optimization 6: *Interval 5. 40'000 calls. Duration Cost*

This section analyses the same scenario as optimization 4 using the process duration cost.

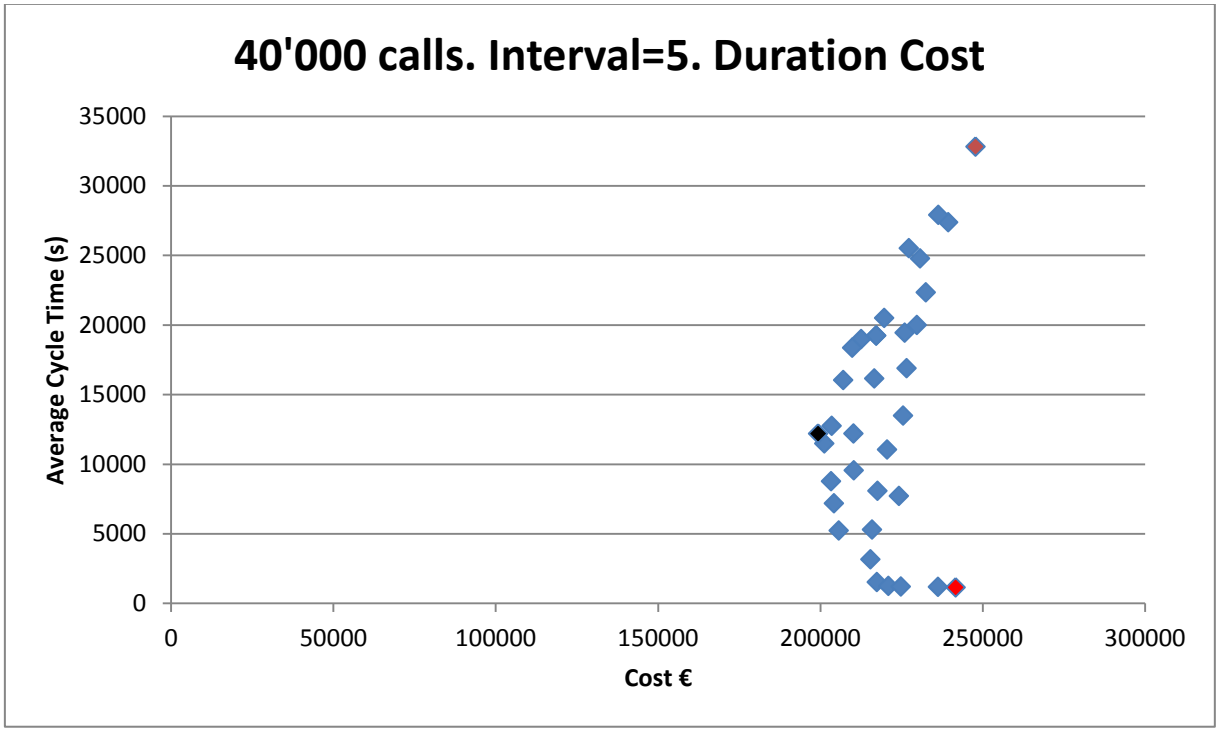

Figure 24: Optimization Results with Interval 5, 40'000 Calls & Duration Cost

Lowest CT:	**1157 s**	**100-90-220**
Lowest Cost:	199279€	60-60-175
Simulation Run Time:	182 s	

Table 14: Optimization #6 Results

When optimizing the stormy season scenario the results are similar to the normal scenario optimization. Due to the fact that the starting position has fairly unsuitable resource allocations then optimizations on the cost dimension happen to be such that the overall cycle time of the process also decreases. Based on the current information that if the insurance claims company did in fact deploy an overtime recovery plan to deal with unhandled insurance claims then actually the most cost-optimal resource allocation would be 60-60-175 (black dot). This would mean an expenditure of 199300€ per week for resources and an average customer would have to wait around 12200 seconds (about 3.4 hours) to get his/her claim processed.

The path of this hill-climbing algorithm can be found in Appendix B.

Optimization 7*: Pareto Frontier*

The previous six optimizations have already shown that hill-climbing can be applied to business process optimization. However, there is no proof that the results I am achieving are the most cost and time-efficient. To investigate I simulated the optimization 6 scenario using 200 random resource allocations. The results are below:

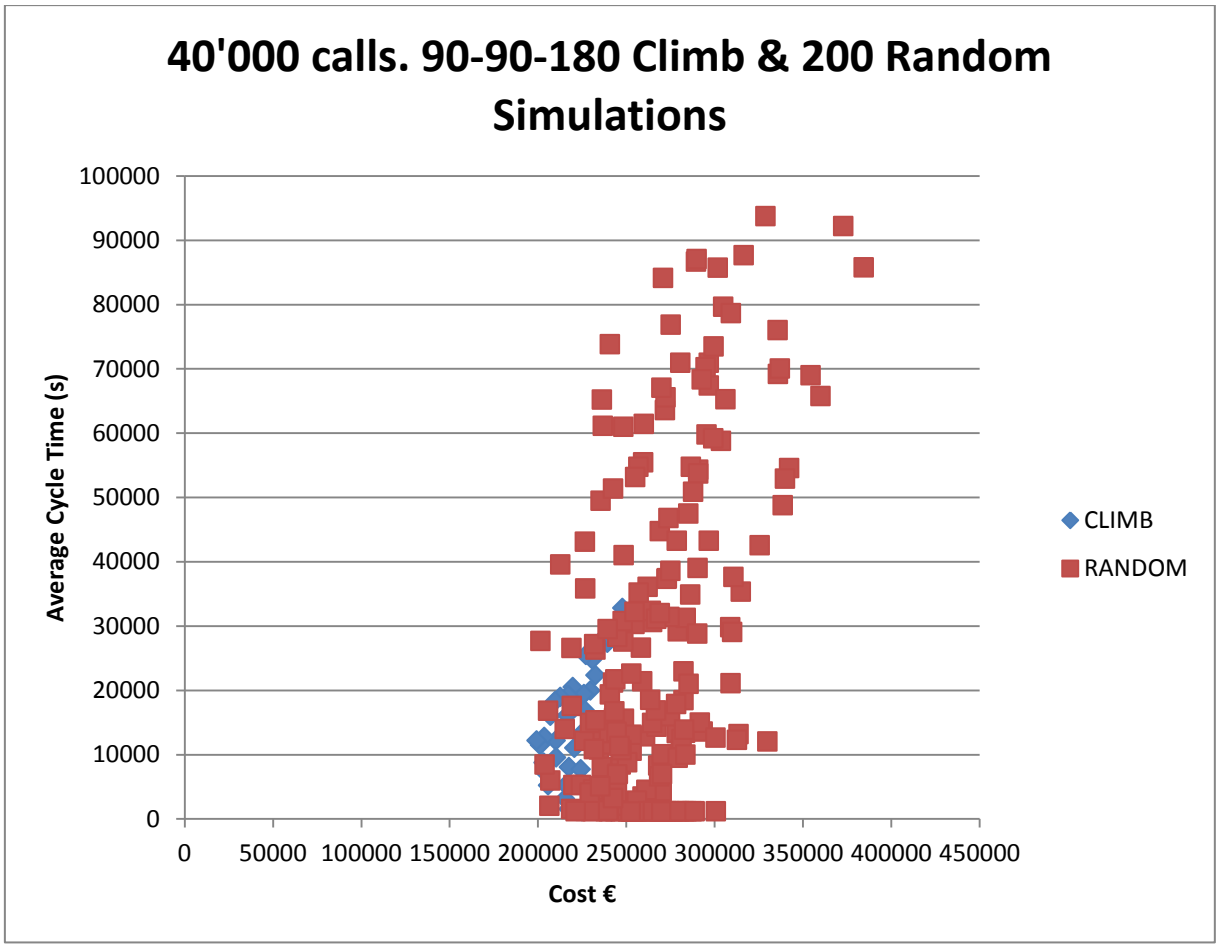

Figure 25: 90-90-180 Climb vs. Random Simulations

	Result	Allocation
Climber Lowest CT	1157	100-90-220
Random Lowest CT	**1155**	85-106-247
Climber Lowest Cost	**199279.3**	60-60-175
Random Lowest Cost	201397.1	51-56-147
Climber Optimization time	182 s	
Random Simulation Time	**117 s**	

Table 15: 90-90-180 Climb vs. Random Simulations Results

As one can see then randomly it is possible to find resource allocations that perform better on cycle time and almost as good on cost. Simulating 200 random points took 117 seconds which a third faster than the time spent by the hill-climbing algorithm. Altogether using a hill-climbing algorithm is not justified if trying random resource allocations is more effective and faster.

As demonstrated earlier then the starting point of the hill-climbing is the most important optimization parameter. This lead me to start investigating what features about a starting point are important in order to achieve better results. My first approach was to simulate the process once and then draw conclusions regarding the best starting point based on resource utilization. During the experiments with resource utilization I soon came to an understanding that it is best to start from the borderline positions: minimum and maximum allowed resource allocation. Moreover, starting from the minimum position has proven to be the most effective when one is interested in finding the most *Pareto efficient* resource allocations in the allowed allocation ranges. An example of the 40'000 call per week scenario starting from the minimum values 50-50-100 is seen in Table 16.

1	**Starting Point**	**MIN(50-50-100)**
2	Melbourne Agents (start-min-max)	50-50-150
3	Sydney Agents (start-min-max)	50-50-150
4	Claim Handlers (start-min-max)	100-100-250
5	Climbing Interval	5
6	Use Simulation Replay	FALSE
7	Call Count	40000

Table 16: 40'000 Calls with Minimum Starting Point Optimization Parameters

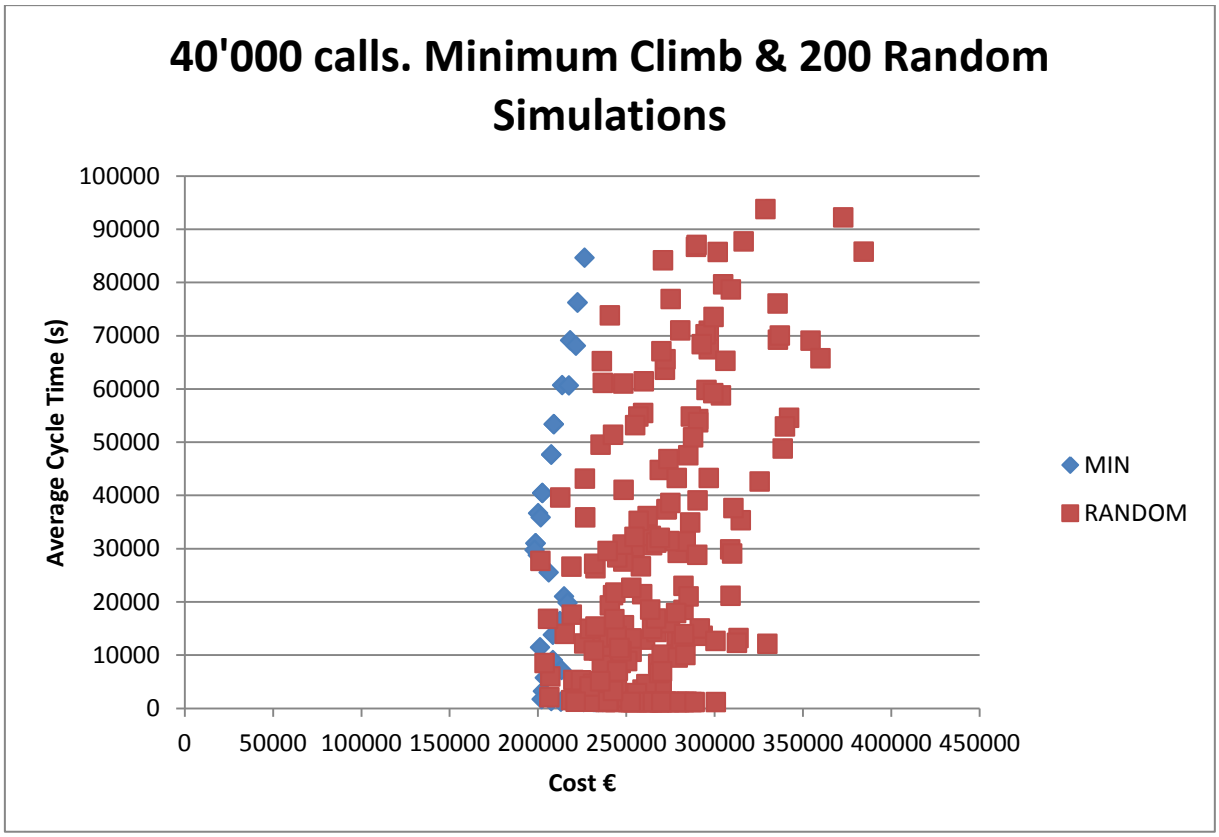

Figure 26: 40'000 Calls with Minimum Starting Point Results

This result displays that starting at the minimum resource allocation values gives high-quality results when one is interested in cost-time trade-off and the *Pareto frontier*. As it can be seen from the graph then there are only a few random points that are more *Pareto efficient* than the ones discovered by the hill-climbing algorithm. This is a noticeable improvement compared to the hill-climbing results that were achieved by starting from the default allocation values.

From the above graph it is seems that the minimum value start point has managed to find the most cost-efficient points but below in the below graph one can see the same graph zoomed in into the 1150-1350 second cycle time area. Here it becomes evident that the hill-climbing algorithm has terminated when climbing near-optimal cycle time resource allocations.

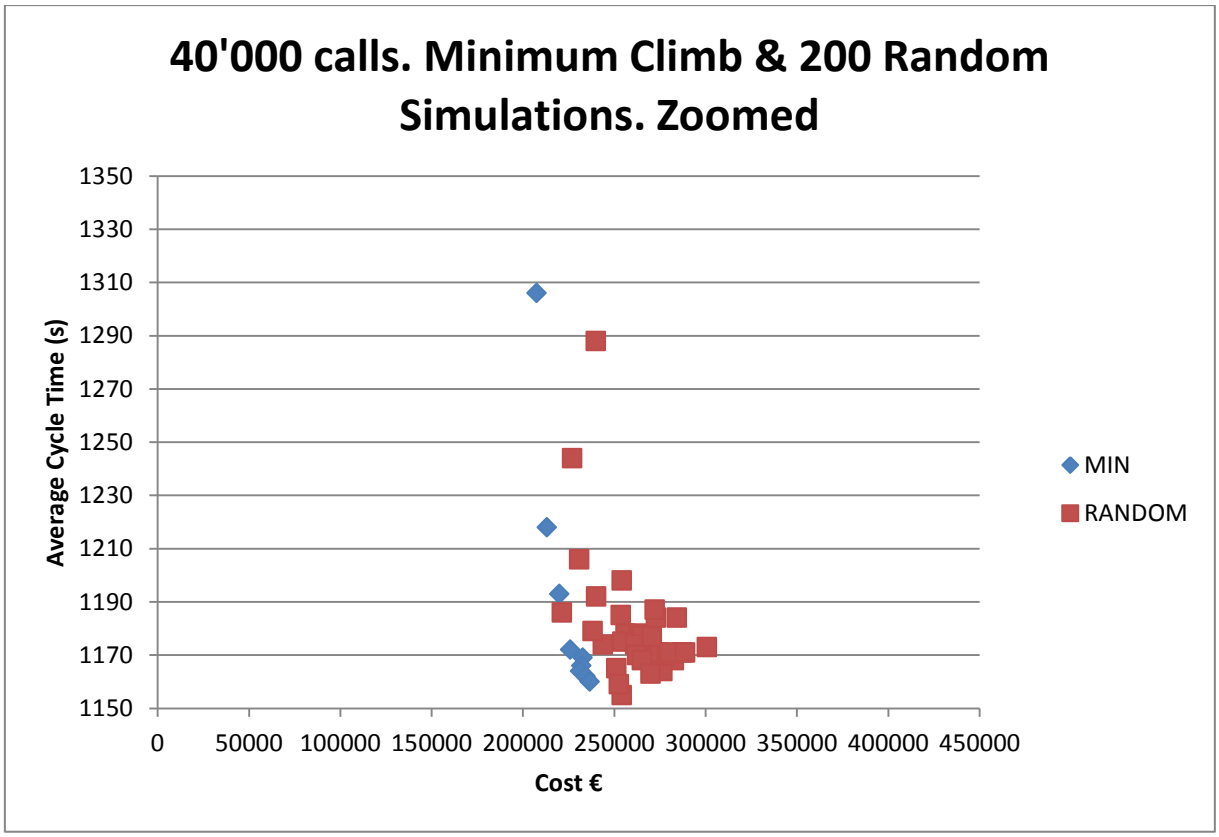

Figure 27: 40'000 Calls with Minimum Starting Point Results Zoomed

I believe that variance in simulation results can be blamed for the few random allocations achieving better cycle time and the early termination of the hill-climbing algorithm.

Interestingly, the same minimum starting position approach works on the 18'000 call per week scenario.

1	Starting Point	MIN(20-20-70)
2	Melbourne Agents (start-min-max)	20-20-100
3	Sydney Agents (start-min-max)	20-20-100
4	Claim Handlers (start-min-max)	70-50-150
5	Climbing Interval	5
6	Use Simulation Replay	FALSE
7	Call Count	18000

Table 17: 18'000 Calls with Minimum Starting Point Optimization Parameters

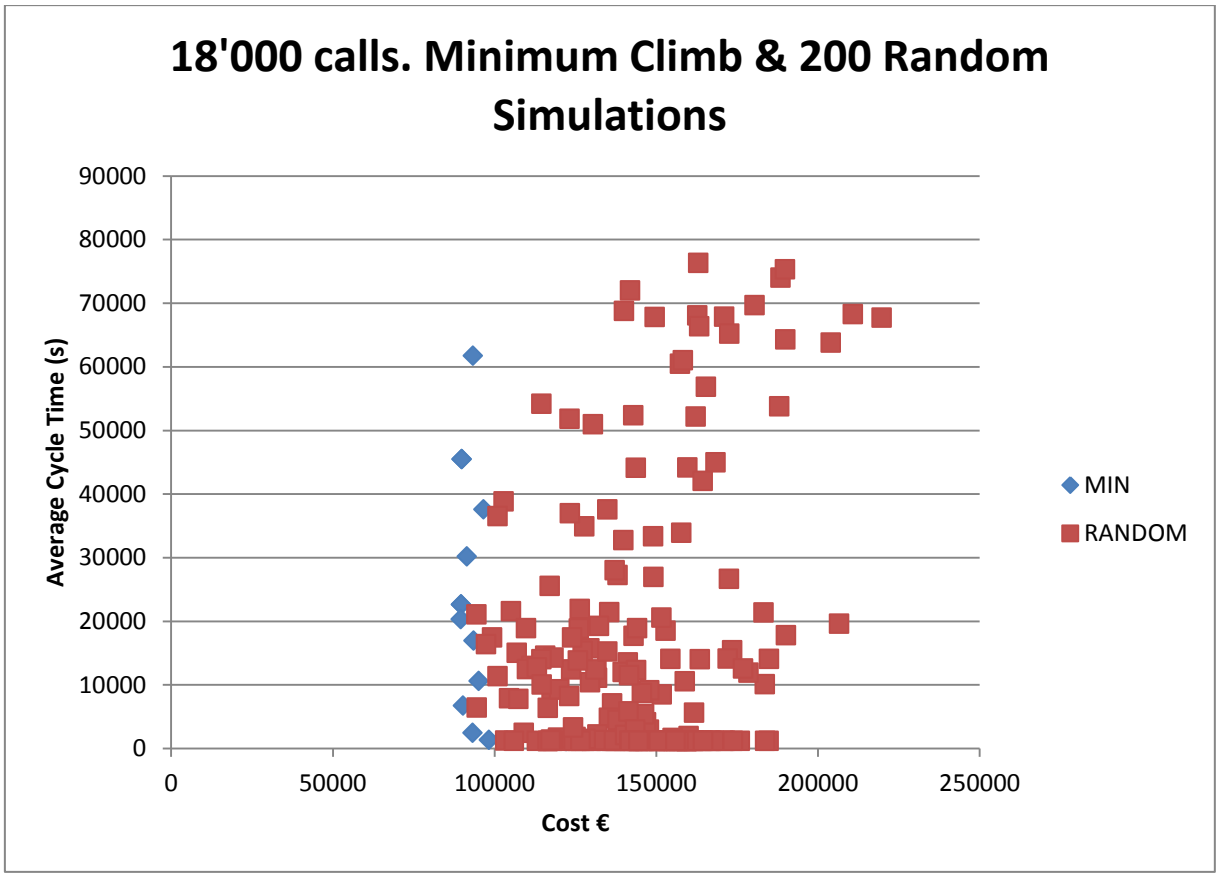

Figure 28: 18'000 Calls with Minimum Starting Point Results

	Result	Allocation
Climber Lowest CT	1154	40-50-110
Random Lowest CT	**1153**	41-51-108
Climber Lowest Cost	**89572.71**	30-30-85
Random Lowest Cost	94440.79	22-25-71
Climber Optimization time	**50 s**	
Random Simulation Time	64 s	

Table 18: 18'000 Calls with Minimum Starting Point Results

The result of using the hill-climbing algorithm starting from minimum values proves to be even more efficient on the normal scenario. It managed to find the best cost and almost the best cycle time allocations. After zooming in close on the lower cycle time region then it appears that the hill-climber only failed to outperform a single random allocation. This is an improvement compared to the stormy 40'000 call a week scenario where multiple random allocations outperformed the hill-climber on cycle time. Although the situation is better with the 18'000 call scenario I still believe that simulation variance is limiting the efficiency of the hill-climbing algorithm.

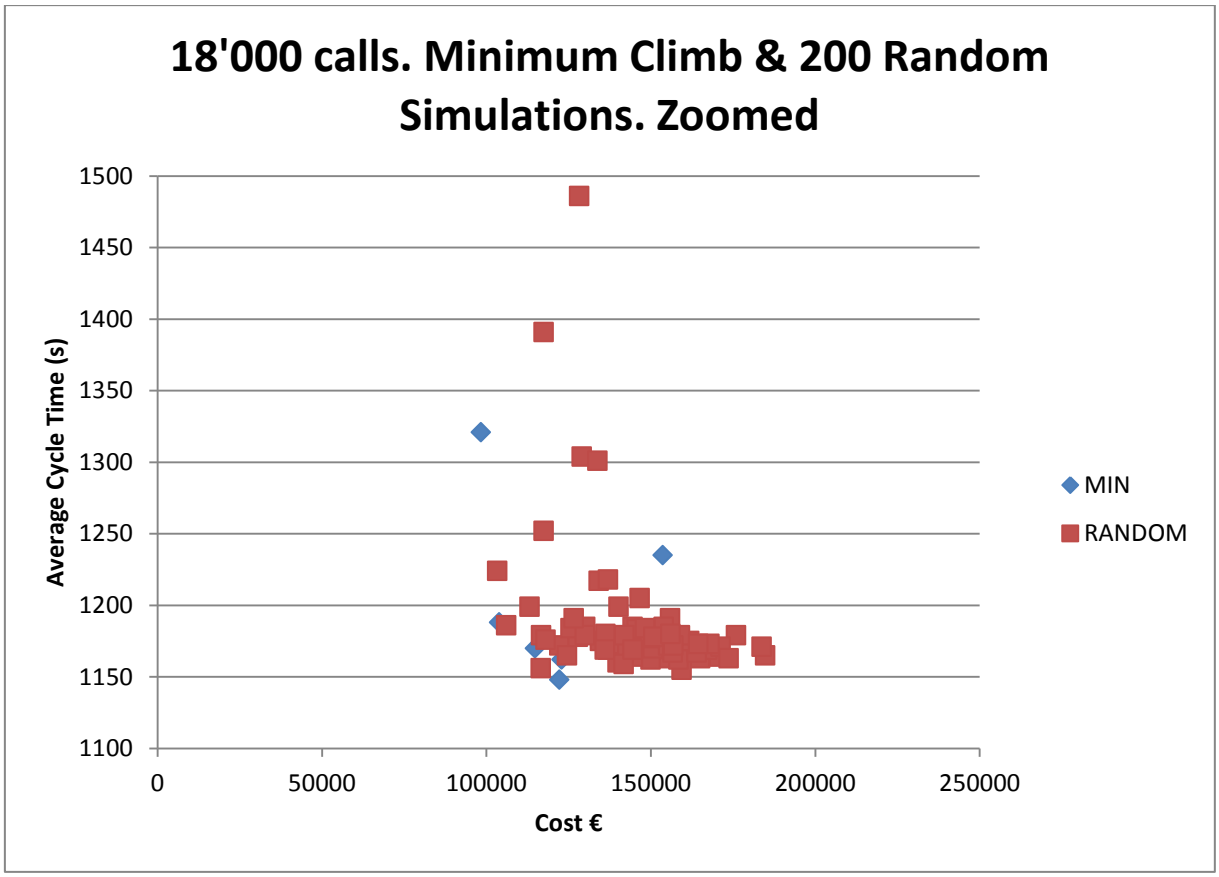

Figure 29: 18'000 Calls with Minimum Starting Point Results Zoomed

Optimization 8: *Variance Removed. Minimum Starting Point. 40'000 calls*

I wanted to prove my assumption that simulation variance is the reason why starting at the minimum allowed resource allocations appears not to deliver a perfect *Pareto frontier*. To do this I implemented the *Simulation replay* functionality that enables the hill-climber to run one simulation and store its arrival rates, gateway selections and process durations. All this was done so that it could be possible to re-run the same simulation with different resource allocations and therefore making them directly comparable.

Below are the results of applying the same stormy season simulation for all resource allocations.

1	**Starting Point**	**MIN(50-50-100)**
2	Melbourne Agents (start-min-max)	50-50-150
3	Sydney Agents (start-min-max)	50-50-150
4	Claim Handlers (start-min-max)	100-100-250
5	Climbing Interval	5
6	Use Simulation Replay	TRUE

7	Call Count	40000

Table 19: Optimization #8 Parameters

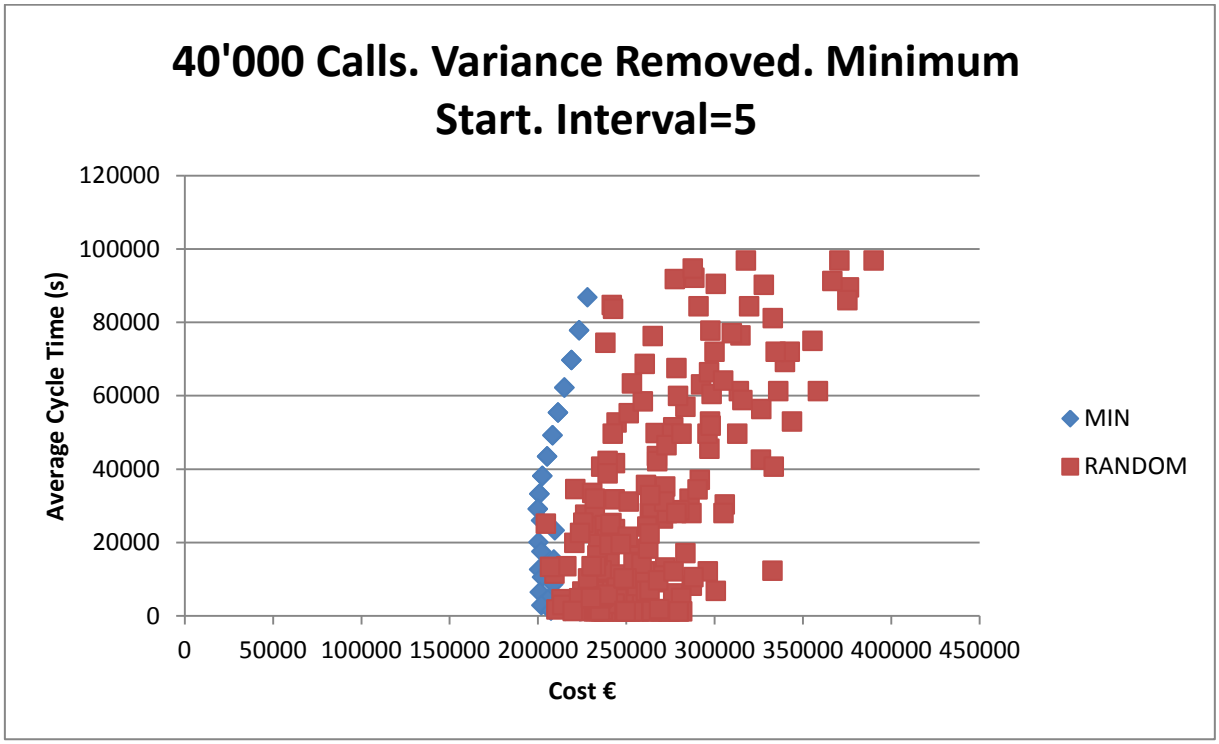

Figure 30: 40'000 Calls, Variance Removed, Minimum Start & Interval 5 Optimization Results

	Result	Allocation
Climber Lowest CT	**1172.534**	120-110-250
Random Lowest CT	1172.613	112-96-249
Climber Lowest Cost	**199973.4**	50-50-145
Random Lowest Cost	204508.5	54-60-154
Climber Optimization Time	324 s	
Random Simulation Time	**128 s**	

Table 20: Optimization #8, Interval 5 Results

Using the same simulation data for all allocations seems to do exactly what I needed it to do. Now the hill-climbing path is much smoother compared to the *Pareto frontier* achieved by the previous optimization that took variance into account. Moreover, when all the variance is removed it can be seen that the hill-climbing algorithm moves along the perfect *Pareto frontier* and no random allocation can outperform it in any dimension. Even on the zoomed graph (Figure 31) the low cycle time zone the points identified using hill-climbing are performing better than the same cost allocations found by random.

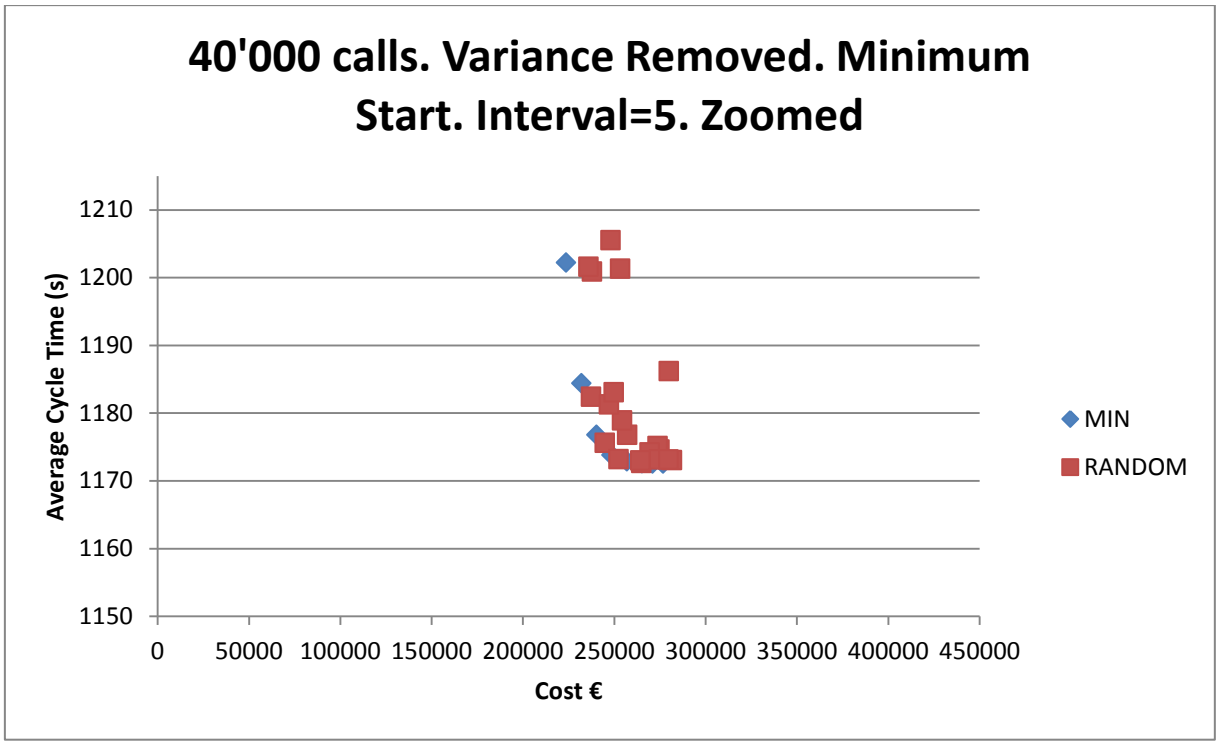

Figure 31: Optimization #8, Interval 5 Results Zoomed

This zoomed graph even shows that reusing the same simulation data removes the largest downside of the regular hill-climbing algorithm: early termination. The graphs in Optimization 7 demonstrated how the hill-climber terminated as soon as it reached almost lowest cycle times. With variance removed the hill-climber continues even when adding an additional resource improves the average cycle time by fractions of a second. For example, the last three average cycle times in this 40’000 call hill-climbing scenario were 1172.596, 1172.543 and 1172.534 seconds.

The hill-climbing path of this optimization can be found in Appendix C.

Using the same simulation data for all optimization steps means that it is finally possible to use lower climbing intervals. Previously it was necessary to use an interval of 5 as this was the only way to avoid invalid terminations early in the optimization process. But now the variance has been nullified using other measures and therefore it is possible to use a smaller climbing interval. An example of using a climbing interval of 1 for the 40’000 call scenario is given below.

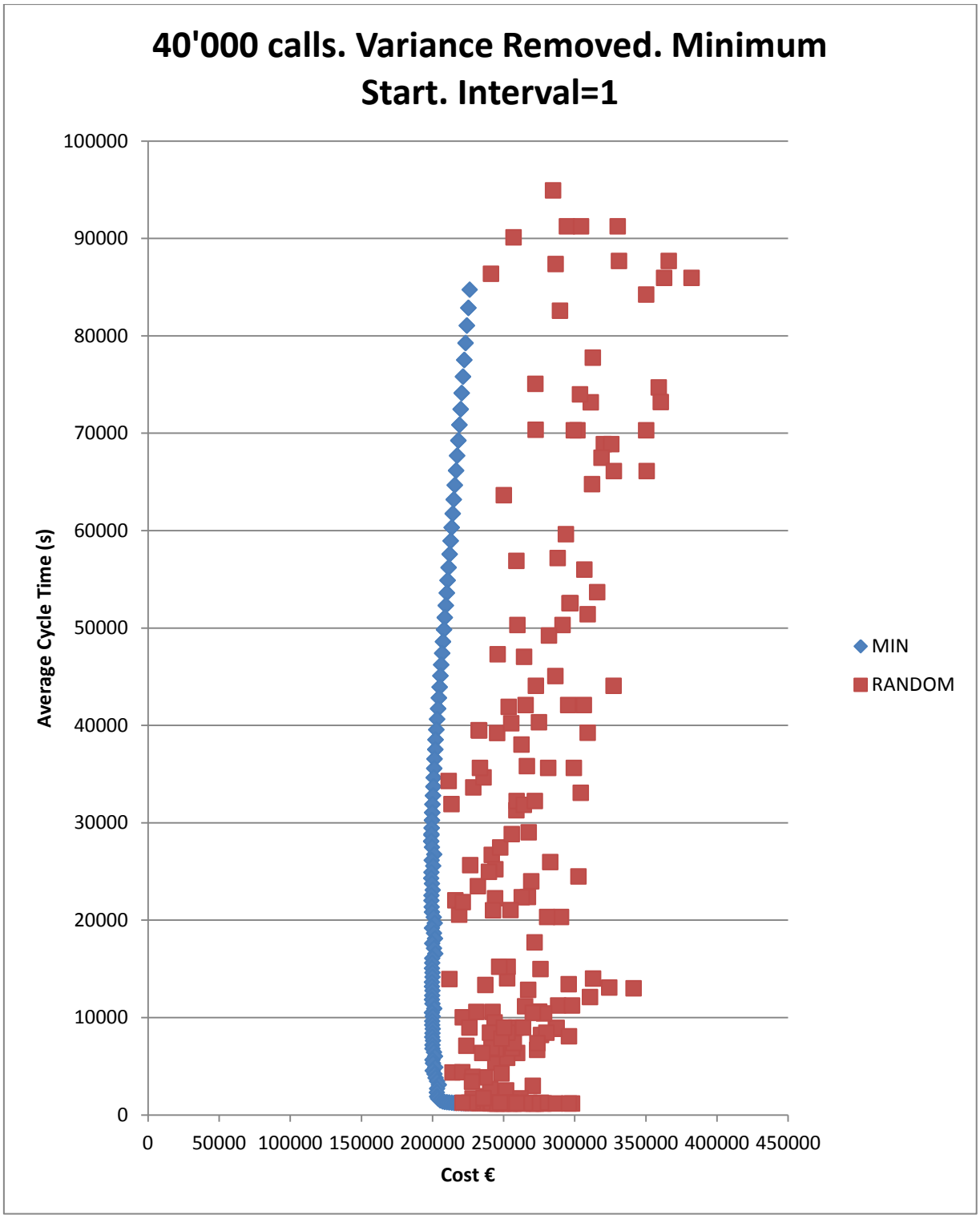

Figure 32: Optimization #8, Interval 1 Results

Figure 32 demonstrates the full path of the optimization process using hill-climbing. Each small fluctuation in the cost and cycle time trade-off is seen and it is easy for a business analyst to choose the trade-off that he/she believes is the most justified. Moreover, in the previous optimizations some assumptions had to be made in order to figure out all the

points on the *Pareto frontier* but with a climbing interval of 1 there is nothing more to guess.

All in all removing the variance and starting from the minimum resource allocations has proven to be a very efficient technique in determining the *Pareto frontier* of a business process. The amount of guess-work that a business analyst is required to do has been reduced significantly. The user's only main responsibility is to set the allowed ranges for all resources. This task is actually just necessary for the purposes of speeding up the optimization process. If he/she feels uncomfortable setting the ranges then it is possible to set the minimum allowed resources to 1 and infinity as the maximum. The hill-climbing algorithm will work just as well but it will just take a longer time to reach optimal allocations as the starting point is further apart from optimal resource allocations.

4.2. Multi-Resource Optimization

After demonstrating that a hill-climbing algorithm can be applied to the insurance claims case study I started to wonder if the algorithm is suitable for scenarios where each task can be executed by more than one resource. To attempt such optimization I first had to modify the *Lightning Fast Business Process Simulator* [2] so that it supported assigning multiple resources to a task. Second, I made an additional assumption regarding the insurance claims case study. The multi-resource optimizations in this thesis are done on the assumption that the insurance company has the option to hire junior call centre agents and junior claim handlers instead of or in addition to the current standard employees. Regarding salary and productivity then I made the assumption that junior staff are paid 20% less but they require 20% more time to execute a task. For example, a standard call centre agent gets paid 2000€ a month and requires around 520 seconds to execute the "Register Claim" task. On the other hand, a junior call centre agent gets paid 1600€ a month and requires an average of 624 seconds to register a claim.

Unfortunately, using hill-climbing for multi-resource allocation optimization comes with a few drawbacks. The main problem is that the number of different resources that can be active in the business process is limited. As I pointed out in the 3rd section of this thesis then the amount of possible neighbours for each resource allocation grows exponentially when adding new resources. This effectively means that in the single-resource optimization each position had up to 26 neighbour allocations that needed to be simulated. However, in the junior worker scenario the amount of potential neighbours for a given allocation is 728.

This problem creates a situation where each hill-climbing step will take around 26 times longer. There are four solutions to this problem:

1. One is to use a very large climbing interval so that the hill-climbing algorithm can cover the whole search space with fewer steps. It will be fast but a lot of information will be lost.
2. The second solution is to use a declining climbing interval so that the optimal cost and cycle time ranges are found quickly and then the optimizer can start searching for a more precise solution in that range. This is useful when the best cost and best cycle time are of interest to us but if the objective is to see the cost-time trade-off on a *Pareto frontier* then such a solution is not optimal.
3. The third solution is to define smaller allowed ranges for each resource so that the optimizer has fewer simulations to do. However, this needs an expert user who is good at estimating resource requirements.
4. The fourth solution is to be very smart about the position the optimizer starts its climbing. Just like with the single resource optimizer I tried different starting positions but in the end I came to a conclusion that setting the start location to the minimum values is yet again the one that creates the best results.

Optimization 9*: Multi-Resource 18'000 Calls*

Below are the results of optimizing the 18'000 calls per week scenario with junior workers also available. I used the simulation replaying strategy to avoid terminations and inconsistencies due to simulation variance.

1	Starting Point	MINIMUM(10-10-10-0-0-0)
2	Melbourne Agents (start-min-max)	10-0-150
3	Sydney Agents (start-min-max)	10-0-150
4	Claim Handlers (start-min-max)	10-0-250
5	Junior Melbourne Agents (start-min-max)	0-0-150
6	Junior Sydney Agents (start-min-max)	0-0-150
7	Junior Claim Handlers (start-min-max)	0-0-250
8	Climbing Interval	10
9	Use Simulation Replay	TRUE
10	Call Count	18000

Table 21: Simulation #9 Parameters

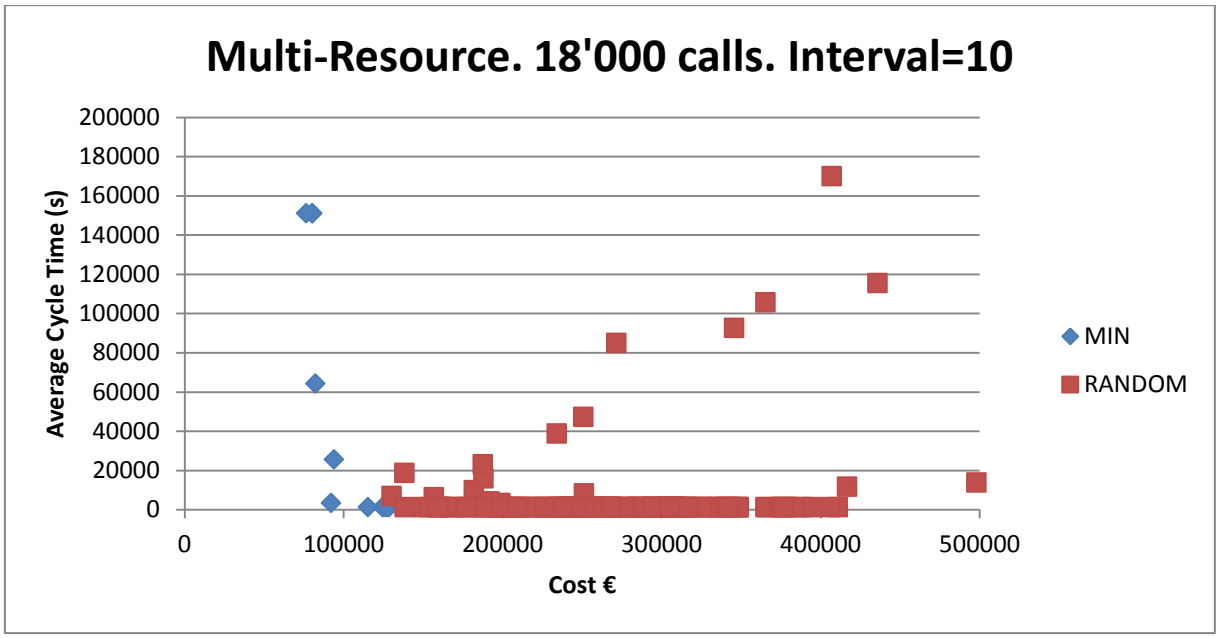

Figure 33: 18'000 Calls Multi-Resource Optimization

	Result	CH	Jun CH	MCA	Jun MCA	SCA	Jun SCA
Climb Best Cost	**76459,06**	10	20	0	10	0	10
Climb Best CT	**1175,359747**	130	10	0	60	0	60
Random Best Cost	**130287,8594**	36	50	12	16	64	59
Random Best CT (82 times)	**1175,359747**	229	81	108	67	108	41
F	**971 s**						
Random Run Time	**92 s**						

Table 22: 18'000 Calls Multi-Resource Optimization Results

The results of the multi-resource optimization are rather interesting. The most interesting is probably the resource allocation that the hill-climbing algorithm identifies as having the lowest cycle time. It consist of 130 Claim Handlers, 10 Junior Claim Handlers, 60 Junior Melbourne Call-Centre Agents and 60 Junior Sydney Call-Centre Agents. It is very fascinating that Junior Call Centre Agents are preferred to the standard ones while standard Claim Handlers are preferred to the junior ones. Both of them were set to be 20% cheaper and 20% slower so there is no straightforward answer why such an optimization is preferred. It might have something to do with call centre agents having lower resource

utilization and therefore it is cheaper for the insurance company to have junior staff waiting for calls instead of senior staff. However, I am not capable of proving my hypothesis. On the other hand, this junior staff situation shows the strength of the hill-climbing optimization approach. The optimizer discovers by itself that junior call centre agents are more cost and cycle time efficient than standard call centre agents. Moreover, the hill-climbing algorithm manages to find the *Pareto frontier* and no random points could be found that are more *Pareto efficient* than the ones found by the hill-climber.

On the downside the hill-climbing process took a very long time to run compared to the random simulation. The optimizer actually did only 13 hill-climbing steps but an average step took 75 seconds. The performance of the hill-climber is rather slow and very close to the situation where trying random resource allocations can produce almost as good results in much less time. However, for this case study the minimum starting point seems to deliver the perfect *Pareto frontier* without the need for simulating all possible resource allocations. As simulating all possible combinations would take days instead of 971 seconds then using the hill-climbing algorithm is still justified.

Another negative side of the results is that the climbing interval is very large and as a result the gaps between different results are huge. This makes it very difficult to understand the time and cost trade-off and how it changes over the course of the optimization process.

The hill-climbing path of this optimization can be found in Appendix D.

Optimization 10: *Multi-Resource 40'000 Calls*

After optimizing the multi-resource allocation with the normal scenario I was also interested to see how the optimizer would handle situations where the process is under heavy load and there is a shortage of staff compared to the default allocation. Moreover, I changed the climbing interval to 5 to have a more detailed view of the climbing steps.

The results of optimizing the 40'000 calls per week multi-resource scenario are below:

1	Starting Point	MINIMUM(10-10-10-0-0-0)
2	Melbourne Agents (start-min-max)	10-0-150
3	Sydney Agents (start-min-max)	10-0-150
4	Claim Handlers (start-min-max)	10-0-250

5	Junior Melbourne Agents (start-min-max)	0-0-150
6	Junior Sydney Agents (start-min-max)	0-0-150
7	Junior Claim Handlers (start-min-max)	0-0-250
8	Climbing Interval	5
9	Use Simulation Replay	TRUE
10	Call Count	40000

Table 23: Optimization #10 Results

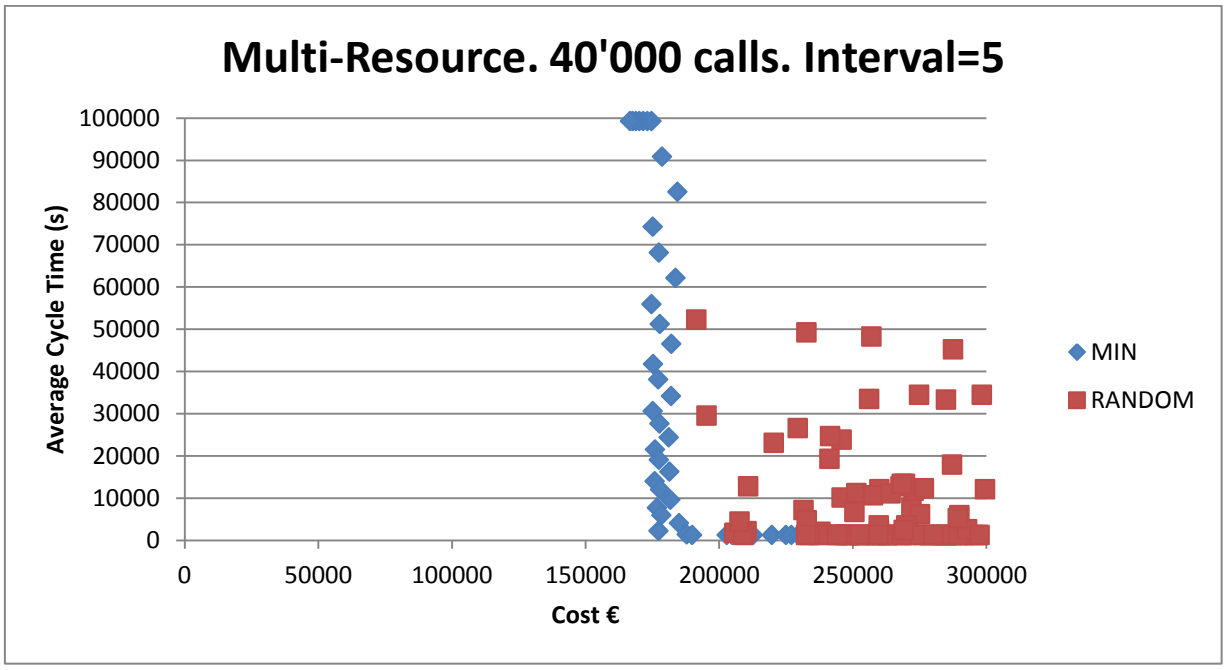

Figure 34: 40'000 Calls Multi-Resource Optimization

				Allocation			
	Result	CH	Jun CH	MCA	Jun MCA	SCA	Jun SCA
Climb Best Cost	**166534,1**	10	65	0	25	0	25
Climb Best CT	**1181,524**	250	15	0	105	0	100
Random Best Cost	191494	80	43	27	27	6	38
Random Best CT (2times)	**1181,524**	248	167	46	61	9	103
Optimization Run Time	21754 s						
Random Run Time	**87 s**						

Table 24: 40'000 Calls Multi-Resource Optimization Results

The results of running the optimization algorithm on the storm season scenario are yet again fascinating. The optimizer manages to decrease the average cycle time by more than

10 times without making the process more costly. Moreover, in the storm season scenario junior call centre agents are still preferred to standard call centre agents. There also seems to be some usage of the junior claim handlers but they seem to be used when the maximum number of standard claim handlers is reached. Also, the hill-climbing algorithm managed to reach the best possible cycle time starting from very low resource allocations. I do not know if this is specific to the insurance claims case study but the hill-climber did a total of 83 steps during this optimization and still managed to reach the lowest cycle time without getting lost in the search space. It very interesting that although hill-climbing is prone to terminate in a local optimum then for this case study it does not appear to be happening.

On the downside, this hill-climbing algorithm took over 6 hours to run. This means that it is reaching the time requirements that genetic algorithms need to optimize business processes. However, genetic algorithms are not capable of finding the *Pareto frontier* and displaying the cost-time trade-off that is so clearly seen on the graph below.

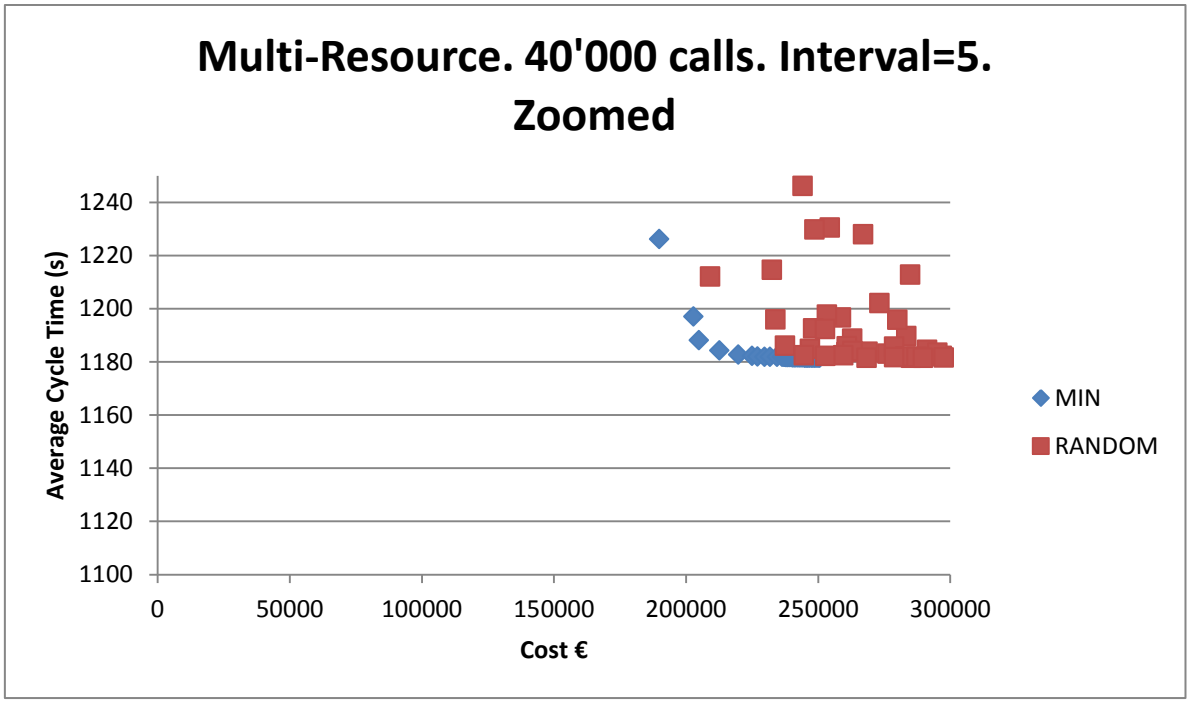

Figure 35: 40’000 Calls Multi-Resource Optimization Zoomed

Figure 15 shows that the hill-climber is achieving almost identical average cycle times (1180-1200 seconds) but the costs are in the range of 190’000 to 250’000. This means that in order to gain around 1% in cycle time one must invest an additional 20% of money/resources. There might be some business situations where such investments are justified but probably not in an insurance company. Figure 15 also shows that the hill-climbing algorithm finds cost-time trade-offs that are very difficult to get close to using

random simulation. For example, most random allocations require more than 225’000 € to reach a cycle time of 1180 seconds while the hill-climbing algorithm achieves that with under 200’000 €.

Altogether optimization of multi-resource processes has proven to be a success using hill-climbing algorithms. Just like single-resource optimization it makes sense to start climbing from the minimum resource allocation and from there the climbing algorithm should follow its path on the *Pareto frontier*. The only downside of multi-resource optimization is that it is very time consuming if one is not ready to sacrifice the climbing interval size. This is caused by the fact that multi-resource simulations usually involves a higher number of resources and this increases the number of neighbours each resource allocation can have.

5. Conclusion

In this thesis a hill-climbing algorithm was used to optimize resource allocations in a business process. The overall results were promising and showed that hill-climbing can be used in resource optimization problems.

The most important discovery was that the starting point plays an important role in the overall results of the optimization. Based on the case study investigated it become evident that in order to find the *Pareto frontier* it is advisable to start hill-climbing from the minimum allowed resource allocation values. Starting from the minimum values manages to determine the *Pareto frontier* as well as find the most cost-efficient and cycle time efficient resource allocations. This thesis also demonstrated that a hill-climbing algorithm is capable of finding an optimal resource allocation for a complex business process in a few minutes. This is much faster than other optimizers based on genetic algorithms. Third, I managed to prove that a hill-climbing algorithm can be used to optimize business processes where multiple resources are capable of executing the same task. Fourth, a methodology was introduced which removes the variance in business process simulations without making them unrealistic. This involves running one simulation and recording the arrival rates, gateway choices and task durations and reusing them for all the subsequent simulations. Such an approach was necessary to compare different resource allocations without having to simulate each allocation multiple times.

On the largest remaining roadblocks of the proposed resource optimization method is that the amount of various resource types in the business process should be 4 or lower. This is caused by the number of possible neighbour resource allocations, which grows exponentially. For instance, an optimization with three resources requires a maximum of 26 process simulations for each step whilst six resources require a maximum of 728 process simulation at each step. Designing optimization heuristics that would limit this combinatorial explosion is a clear direction for future work.

Another avenue for future work is to experiment with other optimization heuristics such as simulated annealing. Moreover, it might make sense to use queuing networks instead of simulation to compare the KPIs of different resource allocations. Using queuing networks would remove the need for the simulation replaying technique and would create an more realistic comparison between different allocations. Third, increasing hill-climbing intervals should be implemented. Decreasing intervals were used in this thesis to locate more

optimal resource allocations after first climbing with a large interval. On the other hand, increasing intervals could be used to avoid getting stuck in a local optimum, which is usually the biggest drawback of hill-climbing implementations. This would mean that after the hill-climbing algorithm fails to improve on a performance measure it would start comparing the current allocation to neighbours further away from the current position. Fourth, a history-based evaluator should be implemented that can blacklist some resource allocations based on previous hill-climbing steps. For instance, in the multi-resource allocation if one resource has been zero for three iterations in a row then it could be decided that this resource is inefficient and should be ignored for the remainer of the optimization. Such a situation would reduce the amount of potential neighbour resource allocations significantly and therefore speed up the simulator. Ignoring one resource out of six would reduce the amount of potential neighbours from 728 to 242. A fifth future implementation could be adding part time worker support. This means that the company can hire full-time workers whose hourly wage is lower but have to be paid for all the hours they are at work. In addition or instead part time workers could be hired who are paid more but only for the time they are actually working. Such an optimizer could be useful in businesses where the business process is seasonal or includes short duration specialist tasks.

6. Automatiseeritud ressursside optimiseerimine äriprotsessides

Magistritöö (30 EAP)

Riivo Roose

Resümee

Äriprotsesside juhtimine on organisatsiooni äriprotsesside muutmine produktiivsuse tõstmise või madalamate kulude saavutamiseks. Igas ettevõttes on väärtuse loomise protsess, mis tavaliselt koosneb inimestest, kapitalist ning informatsioonist. Pidevas muutuses olev nõudluse ja pakkumise vahekord tekitab olukorra, kus tihti on vaja äriprotsesse muuta, et säilitada oma konkurentsieelis ja/või kasumlikkus. Äriprotsesse võib parandada kas protsessi struktuuri korrigeerides või muutes ressursside jaotust. Struktuuri saab muuta näiteks väärtust mitte lisavaid ülesandeid vähendades. Ressursside puhul saab palgata juurde inimesi või muuta töötajate töökohustusi. Näiteks jõulude ajal võib logistikafirma palgata juurde kullereid, et kõik pakid õigel ajal kohale toimetada.

Äriprotsesside analüüsimiseks on loodud nii erinevaid modelleerimiskeeli kui ka tarkvaralahendusi. Modelleerimiskeele abil on võimalik äriprotsessi kujutada kõikidele osapooltele arusaadavas vormis. Tarkvara abil on modelleeritud äriprotsesse võimalik simuleerida, saades väljundiks protsessi tulemuslikkuse näitajad. Simuleerimine aitab analüütikutel hinnata erinevate muutuste mõju äriprotsessile. Kahjuks on võimalike optimiseeringute suure arvu tõttu simuleerimise abil protsessi parandamine väga ajanõudev tegevus. See sunnib analüütikuid probleemi lahendama katse-eksitus meetodi abil ning parima võimaliku lahenduse leidmine pole garanteeritud.

Automatiseeritud äriprotsesside parandamist on uuritud ning parimaid tulemusi on saavutatud geneetiliste algoritmide abil ressursside jaotuse muutmisel. See protsess on väga aeganõudev ning tihtipeale suudetakse optimaliseerida ainult ühe tulemuslikkuse näitaja suhtes. Samas on äris tihti vaja teha kompromisse aja ning kulu suhtes. Näiteks viie miljoni euro investeerimine protsessi kiiruse vähendamiseks neljakümne protsendi võrra on tihti rohkem õigustatud kui samasse protsessi viiekümne miljoni euro investeerimine, mis toob kaasa ainult kuuekümne protsendilise arengu.

Eelnevalt nimetatud probleemide lahendamiseks on käesoleva magistritöö eesmärgiks olnud arendada automaatne äriprotsessi ressursside jaotuse optimaliseerija, mis suudab protsessi parandada korraga mitme tulemuslikkuse näitaja suhtes. Tähtsat rolli omas ka optimaliseerija kiirus. Üldised tulemused olid positiivsed ning tõestasid juhtumiuuringu abil, et matemaatilist optimaliseerimistehnikat nimetusega *hill-climbing* on võimalik rakendada äriprotsessi resursside jaotamise probleemi lahendamisel.

Antud magistritöö esimene osa selgitab äriprotsesside juhtumist, sellega kaasnevaid väljakutseid ning lahendusi. Teises osas esitatakse seotud uuringuid ning võetakse kokku praegused lahendused probleemile. Seejärel selgitatakse pakutud lahendust ning selle väljakutseid. Neljandas osas antakse hinnang lahendusele, kasutades esimeses osas selgitatud juhtumiuuringut. Viimases osas võetakse töö kokku ning pakutakse välja suunad edasiseks tööks.

7. References

[1] Florian Niedermann, Sylvia Radeschütz, Bernhard Mitschang: *Deep Business Optimization: A Platform for Automated Process Optimization*. ISSS/BPSC 2010: 168-180 (2010)

[2] M. Abel: *Lightning Fast Business Process Simulator*, Masters Thesis, Institute of Computer Science, University of Tartu (2011)

[3] Wil M. P. van der Aalst, Michael Rosemann, Marlon Dumas: *Deadline-based escalation in process-aware information systems*. Decision Support Systems 43(2): 492-511 (2007)

[4] Jiajie Xu, Chengfei Liu, Xiaohui Zhao: *Resource Allocation vs. Business Process Improvement: How They Impact on Each Other*. BPM 2008: 228-243 (2008)

[5] Andrew Greasley: *Using business-process simulation within a business-process reengineering approach*. Business Process Management Journal, 9(4): 408 – 420 (2003)

[6] Andrew Greasley: *A redesign of a road traffic accident reporting system using business process simulation*. Business Process Management Journal, 10(6):635 – 644 (2004)

[7] Andrew Greasley: *Using process mapping and business process simulation to support a process-based approach to change in a public sector organisation*. Technovation 26:95 – 103 (2006)

[8] W. Kelton, Randall Sadowski, Nancy Swets: *Simulation with Arena*, McGraw-Hill Science/Engineering/Math (2009)

[9] Ka-Leong Chan, Yain-Whar Si, Marlon Dumas: *Simulation-Based Evaluation of Workflow Escalation Strategies*. ICEBE 2009: 75-82 (2009)

[10] Kazuhiro Saitou, Samir Malpathak, Helge Qvam: *Robust design of flexible manufacturing systems using colored Petri net and genetic algorithm*. Journal of Intelligent Manufacturing, 13:339-351 (2002)

[11] Yain-Whar Si1, Veng-Ian Chan1, Marlon Dumas: *Generic Evolutionary Framework for Resource Allocation in Business Processes* (2011)

[12] Pareto Efficiency
http://en.wikipedia.org/wiki/Pareto_efficiency (10.05.2012)

[13] CPN Tools
http://cpntools.org/ (10.05.2012)

8. Appendices

8.1. Appendix A: Hill-Climbing Path of Optimization #2

Predefined starting point. Interval 5. 18000 process instances. Low resource ranges

ClaimHandler	MelbCallAgent	SydCallAgent	Cost	Avg CT
150	90	90	183000	1173
145	85	85	174900	1176
145	85	95	179900	1155
140	80	80	166800	1156
135	75	75	158700	1158
130	70	70	150600	1165
135	70	75	156200	1153
125	65	65	142500	1168
120	60	60	134400	1162
115	55	55	126300	1167
110	50	50	118200	1172
105	45	45	110100	1184
100	40	40	102000	1178
95	35	35	93900	1313
90	30	30	85800	3739
85	25	25	77700	17577
80	20	20	69600	40021
75	20	20	66500	42021
70	20	20	63400	38872
65	20	20	60300	40738
60	20	20	57200	39529
55	20	20	54100	45419
50	20	20	51000	62489

8.2. **Appendix B**: Hill-Climbing Path of Optimization #6

40'000 calls. Interval 5. Duration Cost

ClaimHandler	MelbCallAgent	SydCallAgent	Cost	Avg CT
150	90	90	247773,1042	32823
155	90	85	239418,225	27361
155	85	85	236429,8229	27907
160	90	85	232569,8563	22333
160	85	80	230740,0729	24765
160	80	80	227276,6222	25506
165	85	85	229744,2611	19976
165	80	85	226037,35	19433
165	80	75	219735,5778	20495
165	75	75	217277,4563	19234
165	75	75	217277,4563	19234
170	80	90	226598,4778	16878
170	85	70	216685,6944	16134
165	75	70	212617,4944	18978
170	70	70	209870,9722	18341
175	80	95	225559,25	13484
175	80	65	210206,3611	12181
170	70	65	207053,7535	16038
175	65	65	203548,0313	12732
180	80	90	220630,2542	11031
180	75	60	210393,9938	9542
175	65	60	201324	11480
180	60	65	203325,8285	8753
175	60	60	199279,3333	12206
185	80	85	217622,525	8068
185	80	60	224280,9535	7692
185	65	70	204274,0361	7165
190	80	85	215940,0917	5283
190	65	75	205760,9833	5209
195	75	90	215444,3875	3151

200	80	90	217474,6597	1520
205	85	85	220996,4167	1248
210	90	85	224888,6354	1203
215	95	90	236262,0667	1169
220	100	90	241653,9125	1157

8.3. **Appendix C**: Hill-Climbing Path of Optimization #8

Variance removed. 40'000 calls. Interval 5

ClaimHandler	*MelbCallAgent*	*SydCallAgent*	*Cost*	*Avg CT*
100	50	50	228131,6	86814,12
105	50	50	223330,8	77810,81
110	50	50	218974,5	69646,45
115	50	50	215048,1	62206,74
120	50	50	211517,1	55395,69
125	50	50	208289	49146,18
130	50	50	205315	43395,75
135	50	50	202559,2	38080,31
140	50	50	200979,1	33240,78
145	50	50	199973,4	29088,31
150	55	50	201863,1	25937,66
155	60	50	209632,4	23304,1
160	55	55	200364,7	20069,69
165	60	55	202019,4	17486,91
170	65	55	209098,9	15346,38
175	60	60	200711,6	12617,78
180	65	60	202309	10488,85
185	70	60	208816,4	8590,009
190	65	65	201168,9	6421,499
195	65	70	205751,1	4517,617
200	70	70	202056,4	2870,97
205	75	75	207417,8	1404,339
210	80	80	215414,1	1246,105
215	85	85	223715,1	1202,219
220	90	90	232016	1184,373
225	95	95	240316,9	1176,794
230	100	100	248617,8	1173,822
235	105	105	256918,8	1172,843
240	110	110	265219,7	1172,596
245	115	110	270958,6	1172,543
250	120	110	276697,5	1172,534

8.4. **Appendix D**: Hill-Climbing Path of Optimization #9

Multi-resource. 18'000 calls

Claim Handler	*Junior Claim Handler*	*Melb. Call Centre Agent*	*Junior Melb. Call Centre Agent*	*Sydney Call Centre Agent*	*Junior Sydney Call Centre Agent*	*Cost*	*Avg CT*
10	0	10	0	10	0	149214,7125	705096,4
20	10	0	10	0	10	80390,67167	151016
20	10	0	10	0	10	80390,67167	151016
30	20	0	20	0	20	82220,0925	64226,08
10	20	0	10	0	10	76459,06	151029,5
40	30	0	20	0	30	93906,48	25572,15
50	40	10	30	10	40	92224,55722	3334,889
60	50	20	40	10	50	115270,2014	1178,605
70	60	10	50	0	60	124614,2283	1175,889
80	50	0	60	0	60	124859,05	1175,665
90	40	0	60	0	60	126123,9619	1175,512
100	30	0	60	0	60	127388,8739	1175,411
110	20	0	60	0	60	128653,7858	1175,369
120	20	0	60	0	60	134978,3456	1175,361
130	10	0	60	0	60	136243,2575	1175,36