Aalto University
School of Science
Degree programme in Engineering Physics and Mathematics

Scheduling of Genetic Analysis Workflows in Grid Environments

Bachelor's Thesis
29.4.2015

Arttu Voutilainen

The document can be stored and made available to the public on the open internet pages of Aalto University.
All other rights are reserved.

Aalto University

AALTO UNIVERSITY
SCHOOL OF SCIENCE
PO Box 11000, FI-00076 AALTO
http://www.aalto.fi

ABSTRACT OF THE BACHELOR'S THESIS

Author: Arttu Voutilainen

Title: Scheduling of Genetic Analysis Workflows on Grid Environments

Degree programme: Degree programme in Engineering Physics and Mathematics

Major subject: Systems Sciences | Major subject code: F3010

Supervisor: Prof. Harri Ehtamo

Instructor: Lauri Eronen, PhD, Biocomputing Platforms Ltd.

Abstract:
The purpose of this work is to compare multiple well-known algorithms for workflow scheduling in grid environments, such that execution times, resource requirements and file transfers are taken into account. Six heuristics (Myopic, Min-Min, Max-Min, Suffrage, HEFT, Hybrid) and two metaheuristics (GRASP, Genetic Algorithm) are studied, implemented and tested. An exact solution is investigated but not feasible due to NP-completeness and large scale of the problem. The example workflows and environments used for testing are provided by BC Platforms and represent usual use cases in the field of genetics.

The first workflow consists of 838 jobs in 22 parallel flows of three stages, the last of which is divided into 11-62 parallel lines. The second workflow consists of 19200 jobs in 100 parallel flows of three stages with merging, splitting and shuffling inside the flows. Both are quite regular and relatively computationally heavy compared to file transfer times, but they are still different enough to test different aspects of the schedulers. The three example environments range from one to 16 resources. Also a version of the third environment with heterogeneous processing powers and bandwidths is used.

During testing, the algorithms are evaluated based on both the makespan of the produced schedule and the time the algorithm required to produce it. Schedules are generated for each workflow on each environment, and the schedules for the first workflow are confirmed using a simulator provided by BC Platforms. The heterogeneous version of environment is used to test the effect of the information about the resources. The HEFT algorithm is found to perform best or be very close to best on each test case in both makespan and scheduling time.

Date: 29.4.2015 | Language: English | Number of pages: 23+12

Keywords: scheduling, workflow, grid, cloud, heuristic, metaheuristic, HEFT, genetic algorithm

Contents

1 Introduction **1**

2 Theory and existing research **2**
2.1 Optimization problem and exact solution 2
2.2 Heuristics 4
2.3 Metaheuristics 5
2.4 Scheduling strategies 7
2.5 Comparison of methods 8

3 Research problem and methods **9**
3.1 Example workflows 10
3.1.1 Imputation with pre-phasing 10
3.1.2 BWA/GATK 11
3.2 Example Environments 12

4 Results **13**
4.1 Imputation workflow 14
4.2 BWA/GATK workflow 16
4.3 Heterogeneous resources 17
4.4 The Genetic Algorithm and GRASP 18

5 Conclusions **19**

References **22**

Appendix A Imputation Workflow **24**

Appendix B BWA/GATK Workflow **26**

Appendix C Results - Imputation **28**

Appendix D Results - BWA/GATK **31**

Appendix E Yhteenveto (in Finnish) **33**

1 Introduction

Biocomputing Platforms Ltd. develops software for genetic research data management and analysis. The company is developing a new workflow engine for executing data processing and analysis workflows in distributed computing environments. The purpose of this work is to compare different scheduling algorithms, and select the best fit to be used by the BC Workflow Engine.

The development of DNA sequencing technology has led to a massive increase in the amount of data that is available to researchers (for the two example cases used in this work, the input data sizes are 16 TB and 450 GB). Same thing has happened in many other fields also, increasing the the need of calculation power and data storage. One answer to these problems is *grid computing* (Foster and Kesselman [2003]), where computations are executed in a network of globally distributed computers. Unlike traditional clusters, grids usually consists of heterogeneous resources, meaning that the *execution time* of a task may differ significantly from one resource to another. Grids have also notable (and varying) *communication costs* (file transfer times), since two processing units can be located for example on different continents, and connected only by the Internet. These facts make scheduling even more important and also harder.

Optimal workflow scheduling is an NP-complete problem (Ullman [1975]), so finding the perfect solution is not computationally viable in all cases. However, there exists many heuristic and metaheuristic methods designed to find a "good-enough" solution. In this thesis some of those methods will be implemented and evaluated based on the *makespan* (the time from the beginning of first task to the end of the last task) of the schedule they produce as well as the time the method takes to execute (*scheduling time*). Two example workflows are provided by the company to be used for testing: BWA/GATK and Imputation with pre-phasing.

Workflow scheduling can be divided into best-effort and Quality of Service -constraint based scheduling. Best-effort workflow scheduling tries to minimize makespan at all costs. QoS-constraint based scheduling takes into account also other things, such as execution costs, and fairness. For the purpose of this thesis, only best-effort schedulers are considered. Also some simplifying assumptions about the grid structure are made, and many factors are left out to simplify the scheduling process. Still, the results are general enough to be of use when comparing the algorithms.

2 Theory and existing research

A *workflow* can be defined as a directed acyclic graph (DAG). The graph has vertices, representing tasks, and edges, representing communication between the tasks, with edge weights representing the amount of communication required. The tasks have also execution times and some resource requirements.

The *resources* are represented as a complete graph, where vertices are processing units and edges represent the communication speed between those units. In a grid there might be multiple processing units connected to one disk or multiple disks connected to one processing units, but in this work the processing units and disks are consider as single entities that form the vertices of the resource graph. The communication speeds are assumed to be symmetric, so the graph is undirected. The vextices are associated with processing powers and details about the resource. *Scheduling* is then an operation of mapping (creating a schedule) the workflow to the environment (a set of resources) at a given time. (Yu et al. [2008])

Following the taxonomy by Yu and Buyya [2005], a grid workflow system consists of four parts: 1) Workflow design, 2) Workflow scheduling, 3) Fault tolerance and 4) Data movement. This thesis considers only workflow scheduling: the workflow specifications are given by the BC Workflow Engine, which also takes care of fault tolerance and data movement. Data movement is presumed to be peer-to-peer between the resources, and the scheduler architecture is supposed to be centralized with performance-driven scheduling strategy.

Section 2.1 defines workflow scheduling as an optimization problem and argues why it is not feasible to be solved exactly, motivating the need for heuristics. The algorithms implemented for this work are Myopic, Min-Min, Max-Min, Suffrage, HEFT, Hybrid, GRASP and a Genetic Algorithm, as they are described by Yu et al. [2008] with possible improvements from other papers. They are described in Sections 2.2 and 2.3. Sections 2.4 and 2.5 focus on existing results related to scheduling.

2.1 Optimization problem and exact solution

The workflow scheduling problem can be stated formally as follows:

Let Γ be the set of tasks T_i, and Λ the set of directed edges (T_i, T_j). The edge (T_i, T_j) is in Λ if and only if task T_j is an immediate successor of task T_i, and the weight of the edge corresponds to the amount of data that has to be

transferred between the two tasks. The execution of task T_j can only begin after all of its parent tasks have been completed. Then the DAG $\Omega(\Gamma, \Lambda)$ describes the workflow completely.

Let $\mathcal{R}$ be the set of available resources, or calculation nodes, and $\mathcal{R}_i$ be the resource where job T_i shall be executed. Let $t(T_i)$ be the starting time of the job, $p(T_i)$ the time it takes to execute the job T_i on the resource $\mathcal{R}_i$ and $f(T_i, T_j)$ the time it takes to transfer the output of job T_i needed by T_j from resource $\mathcal{R}_i$ to $\mathcal{R}_j$. Let C_R be the number of cores and M_R the amount of memory available on the resource $R \in \mathcal{R}$. Denote by $c(T_i)$ and $m(T_i)$ the number of cores and amount of memory used by the job T_i.

The workflow scheduling problem can then be stated as the following optimization problem:

$$\text{minimize } C_{max} \tag{1}$$

$$\text{subject to: } C_{max} \geq t(T_i) + p(T_i),\ \forall\ T_i \in \Gamma \tag{2}$$

$$t(T_i) + p(T_i) + f(T_i, T_j) \leq t(T_j),\ \forall\ i, j \text{ s.t. } (T_i, T_j) \in \Lambda \tag{3}$$

$$\sum_{\substack{T_i \in \Gamma \text{ s.t. } R = R_i \\ t(T_i) \leq t < t(T_i) + p(T_i)}} c(T_i) \leq C_R,\ \forall t \in [0, C_{max}), R \in \mathcal{R} \tag{4}$$

$$\sum_{\substack{T_i \in \Gamma \text{ s.t. } R = R_i \\ t(T_i) \leq t < t(T_i) + p(T_i)}} r(T_i) \leq M_R,\ \forall t \in [0, C_{max}), R \in \mathcal{R} \tag{5}$$

Here equations (1, 2) mean that we want to minimize the makespan of the schedule. Equation (3) makes sure that no job is scheduled to be started before all of its predecessors have been executed. (4) and (5) make sure that the resource constraints are not violated.

This problem, however, is NP-complete (Ullman [1975]) and as such no polynomial time algorithms is known that would solve it. As the number of jobs and the number of resources can range from just a few to many thousands, an exhaustive search is not a viable option. For example, Binato et al. [2002] defines a disjunctive programming formulation for a Job Shop Problem, which can be seen as a very restricted version of a workflow scheduling problem: it consists of multiple parallel execution lines composed of serial tasks, but the lines are not allowed to interact, and for each task the machine used to execute it is given. The JSP is also NP-hard, even with heavy restrictions. Still, exact methods have been used to solve JSP for small tasks (10 lines x 10 machines) while problems from (15x15) upwards are considered to be too

big to solve exactly, according to Binato et al. The general multi-resource workflow scheduling is a much harder problem and thus heuristics are needed.

2.2 Heuristics

Heuristics are algorithms designed to search the solution space for a "good-enough" solution. All the algorithms described here are *constructive methods* (Blum and Roli [2003]), meaning that they construct a solution according to some rule. These heuristics can be divided into *individual task scheduling* and *list scheduling*, and the latter can be again divided into *batch mode*, *dependency mode* and *dependency-batch mode* algorithms. A batch mode algorithm considers only a set of independent tasks, while dependency mode algorithm takes task dependencies into account when prioritizing tasks.

To facilitate the description of the algorithms, let *Estimated Completion Time* $EAT(t, r)$ be the time at which the task t could complete execution on resource r. It takes into account the file transfer times between the task and its predecessors, the availability of the resource and the execution time of the task on that resource. *Minimum Estimated Completion Time* $MCT(t)$ is the minimum ECT for task t over all resources.

Myopic scheduler is the most simple and naïve of the implemented algorithms. It is an individual task scheduling method: it takes one available task and schedules it to the resource where its ECT is smallest. As the Myopic scheduler considers only one task at a time, it is not very effective. Also, depending on the method used to select the task to schedule at each iteration, Myopic may not always produce the same schedule for a given workflow.

Min-Min, *Max-Min* and *Suffrage* are batch mode algorithms. Like Myopic, they have been initially designed for scheduling independent tasks to a number of resources, so they only take dependencies into account when checking tasks' availability. Instead of just selecting some available task, the Min-Min scheduler picks the task whose MCT is smallest and schedules it to the resource that is able to finish it earliest. This method may produce a schedule with a long makespan if there are lots of short and some very long tasks, as the short ones are scheduled to be executed first and the longest are scheduled to be the last. To address this problem, the Max-Min scheduler selects the task whose MCT is biggest and schedules that task. The Suffrage algorithm schedules at each round the task that has the biggest difference between its MCT and the second-best ECT. The idea is to schedule the task which suffers the most if left un-scheduled.

Heterogeneous-Earliest-Finish-Time (HEFT) (Topcuoglu et al. [2002]) is a dependency mode scheduler. It ranks all tasks based on their properties (execution and file transfer times) and the ranks of their successors. The tasks are then picked in descending order of rank and each task is scheduled to the resource with minimum ECT. In a heterogeneous grid the execution and communication times wary depending on the used resources, so it is not possible to know the exact processing times or file transfer times beforehand. The rank function used in this work is the one described by Yu et al. [2008], which averages the execution and file transfer times over all resources. Other possibilities would include for example using minimum or maximum times. The ranking can be made starting from the last job or the first job (the HEFT algorithm requires the schedule to have a unique last or first job, respectively, but a meta-job for that purpose can be created by the algorithm if the workflow does not have one). Different ranking schemes have been studied by Zhao and Sakellariou [2003]. Their study shows that the ranking method may have a significant impact on the result of the HEFT schedule and that different schemes perform well in different cases.

A hybrid heuristic can be used to combine batch and dependency modes. The Hybrid scheduler (Yu et al. [2008]) first ranks tasks like the HEFT scheduler, after which it partitions them into groups so that all the tasks in a group are independent of each other and have smaller ranks than tasks in the previous group. The groups are then scheduled using a batch-mode algorithm (Max-Min in this work).

2.3 Metaheuristics

Metaheuristics are general problem-solving tools. They combine problem-specific heuristics to search the solution space in a way that allows them to escape local optima in order to find a near-optimal solution. Metaheuristics employ the heuristics to construct possible solutions and also to optimize them with a local search. Instead of doing pure random search, a meta-heuristic uses some intelligence to guide the construction of possible solutions. (Blum and Roli [2003])

The *Greedy randomized adaptive search procedure* (GRASP) is a simple example of metaheuristics. The idea is to construct a viable candidate schedule by iteratively picking (job,resource) pairs from a *Restricted Candidate List* (RCL) and then try to find the *local optima* for that candidate. If the optima is better than the current *global optima*, the newly found local optima is set as the new global optima. This procedure is then repeated for a certain

number of times. To construct the RCL, the (job,resource)-pairs are selected that increase makespan no more than a given threshold, which is usually a function of the smallest and biggest possible increases. The algorithm can be controlled through two parameters: the threshold for the filtering of the RCL and the number of iterations.

A basic GRASP for workflow scheduling problem is described by Blythe et al. [2005]. The version used for this work combines the basic version with a local search described by Binato et al. [2002], where GRASP was applied for the Job Shop Problem. The local search phase tries to find the critical path of jobs and check if swapping two jobs (not depending on each other) on the path would produce smaller makespan. However, unlike in the Job Shop Problem, here a single resource can process multiple jobs at a time, so the operation is harder to implement well. Binato et al. [2002] also describes two improvements, *intensification-enhanced construction* and *Proximate Optimality Principle*, that according to them make the GRASP "slightly better", the former by introducing memory of previous good solutions to the construction phase and the latter by doing local search periodically already during the construction phase. These two improvements are not not implemented in this work.

Genetic Algorithms (GA) are metaheuristic search procedures that try to find a good solution by utilizing the methods that drive evolution in the nature: survival of the fittest, mutation, crossover and reproduction. According to Hou et al. [1994], a genetic algorithm usually consists of three steps: 1) initialization of solution population 2) evaluation of fitness of each solution in the population and 3) generation of new solution population through the use of genetic operations. Steps 2-3 are then repeated either for a certain number of iterations or until some other convergence criteria is fulfilled. To develop a genetic algorithm, they list four major decisions: 1) the representation of a solution 2) the design of the genetic operations used 3) the goodness-of-fit function and 4) the probabilities for the genetic operations.

Hou et al. [1994], after which the GA used here is designed, uses a *height* criteria to facilitate the creation of new candidate schedules, both in the initialization phase and the mutation and crossover stages. The *height* of a job is the maximum of the heights of its predecessors plus one, or zero if it has no predecessors. Hou et al. show that the candidates constructed according to the height criteria (tasks are scheduled in ascending order of height) are valid schedules, but the condition is not necessary: some viable solutions, even good ones, do not respect height ordering. This prevents the usage of other heuristics to construct the initial population (as the schedules

created by the heuristics might not conform to the criteria) and also means that the GA might not be able to construct the optimal solution at all. They also propose and improved version of the criteria, which assigns to each job a random height between the maximum of the heights of its predecessors (plus one) and the minimum of the heights of its successors (minus one). For the workflows used in this work the improved height criteria results in the original heights, due to the structure of the workflows.

With the height ordering, to create an initial schedule the jobs are first grouped by their heights. Then the jobs of each group, starting from the group with height 0 jobs and continuing in ascending order, are scheduled in random order to randomly selected resources. As all jobs of smaller height are scheduled before a job of bigger height, also all of its predecessors must have been scheduled and thus the schedule is valid. The crossover operation picks randomly two schedules and a height, and for each resource swaps the parts of the two schedules where the jobs' heights are bigger than the chosen height. The mutation operation picks randomly two jobs of same height from the schedule and swaps those jobs. After any of these operations the availability times of each job must be recalculated, as the GA is only interested in the distribution of jobs onto the resources and the order of jobs on any single resource.

The optimal parameters for both metaheuristics depend on the workflows and environments, and each case must be studied separately to find the parameters that give good results sufficiently fast. There are two kinds of parameters: some have effect only on the schedules (like the RCL threshold for GRASP and genetic operation probabilities for GA) and the effect is hard to know without evaluating different choices, while others (number of iterations on GRASP and number of generations on GA) affect also scheduling time. Increasing the number of iterations or generations monotonically improve performance but also increase scheduling time. For this work, the parameters were chosen by trial-and-error. For GRASP, a (job,resource)-pair is included in the RCL if the increase in makespan is less than $min\ increase + 0.15 * max\ increase$ and 100 iterations are used. For GA, the mutation probability is set to 0.2, crossover probability is set to 1.0, two best schedules are always copied to next generation, population size is 20 and 100 generations are used.

2.4 Scheduling strategies

Wieczorek et al. [2005] discuss and evaluate different scheduling strategies,

including *full-ahead planning* and partial scheduling strategies *just-in-time (JIT) scheduling* and *partitioning of the workflow* (the partitions can then be JIT-scheduled). In JIT scheduling, a task (or a partition of the workflow) is scheduled only when it becomes available, i.e. all predecessor jobs have been executed. While with a partial scheduling strategy it is easier to take into account the changing situations in the grid usage, a full-ahead strategy gives always at least equally good schedules for the current situation. If the situation changes, the scheduling can be re-done, given that the cost of scheduling is relatively small, as is the case for for example HEFT. They find that the HEFT algorithm performs best with the full-ahead strategy, and that HEFT executes so fast even on the full workflow that there is no need for partitioning.

2.5 Comparison of methods

Wieczorek et al. [2005] compare Myopic, HEFT and a genetic algorithm on the ASKALON grid environment using two workflows, one of which was fully balanced and in that respect similar to the workflows in this work. They also tested two estimates of execution time: all tasks on all resources have the same execution time, or all tasks have individual execution times on all resources, based on historical data. According to their results on the balanced workflow, all three algorithms perform equally well when there were no estimates of execution time. With execution time estimates applied, the makespan of each of the schedules drop, with HEFT's schedule being nearly twice as fast as the no-estimate schedule, alleviating the importance of good estimates. The genetic algorithm performs better than Myopic, but still worse than HEFT. Also it was found to be slower than HEFT by $2-3$ orders of magnitude.

Having introduced the HEFT algorithm, Topcuoglu et al. [2002] compares it with four heuristics not considered in this work: CPOP, DLS, MH and LMT. They found that in general the HEFT scheduler performs best: on 87% of their tests, it produces the best schedule

Braun et al. [2001] compare eleven scheduling methods, including Min-Min, Max-Min and a Genetic Algorithm for scheduling independent tasks on a grid environment. They find that for all of their test cases, the Genetic Algorithm performs better than Min-Min or Max-Min, and that Min-Min outperforms Max-Min. They also include other metaheuristics, such as *Simulated Annealing*, *Genetic Simulated Annealing* and *Tabu search*, but those are shown to be inferior to the Genetic Algorithm.

3 Research problem and methods

The purpose of this thesis is to compare different scheduling algorithms on grid environments on both the makespans of the schedules they produce and their scheduling times. In the previous section, eight scheduling algorithms were described. This section presents the experiments used to evaluate their performance.

To compare the algorithms, they have been implemented in Java by the author. Each algorithm takes as input a file containing the workflow DAG and a file containing the available resources, and as output produces a file containing the generated schedule. Also a utility provided by BC Platforms that simulates the executions of jobs and file transfers, called later the *simulator*, is used to validate some of the schedules. It takes as input files containing the workflow, the resources and the schedule, and produces as output a file containing information about the simulation, including the simulated makespan.

To analyze the performance of the schedulers, BC Platforms provided two example workflows commonly used with their product, as well as three examples of typical execution environments and their resource definitions. The workflows are described in Section 3.1 and the environments in Section 3.2. Each scheduler is used to generate a schedule for both workflows on each environment.

To see how the algorithms behave in more heterogeneous situations, randomized version of one of the environments is also created. This means randomizing the processing speeds, amounts of RAM and network bandwidths. The Impute-workflow is scheduled on both the non-randomized and randomized environments, and the simulator is used to test both schedules on the randomized environment, to get an idea of how much effect the knowledge of the resources has on the real makespan.

As the scheduling times are implementation-specific and vary depending on the computer where the schedulers are run, only their magnitude and relative differences are of interest. Thus they are only run once and the time is measured with the *Unix* program “time” (which measures the actual used CPU time), even though this might not be the most precise timing method. The schedulers are run on a 2 x Intel Xeon E5-2670 with 251 GB of RAM.

3.1 Example workflows

The scheduling algorithms will be tested on two genetic data analysis workflows: BWA/GATK and Imputation. Both share some characteristics: multiple non-interacting lines of jobs are processed in parallel, each line through the workflow has the same number of jobs and both workflows consist of three stages. Differences arise for example in the way jobs are split and merged (Imputation has only splits, while BWA/GATK has also merges), but the biggest differences are in the resource requirements and in the number of jobs: while Imputation has 838 jobs, the BWA/GATK boasts 19200 jobs. The requirements of a job are given as its execution time (on a resource with CPU factor of 1), amount of memory it requires and the number of CPU cores it uses.

3.1.1 Imputation with pre-phasing

In genetics, imputation is used to replace missing parts of a genotype (a set of measured genetic variants for a single individual) using information contained in a reference panel. It allows researchers to cheaply and quickly measure only selected parts of DNA and then fill in the rest, using statistical methods (Howie et al. [2011]). Phasing is the act of inferring the how the genetic variation of an individual is divided between the paternally and maternally inherited chromosomes (Howie et al. [2012]). While it can be done during the imputation process, it is often more efficient to do it separately, as in the example workflow used here.

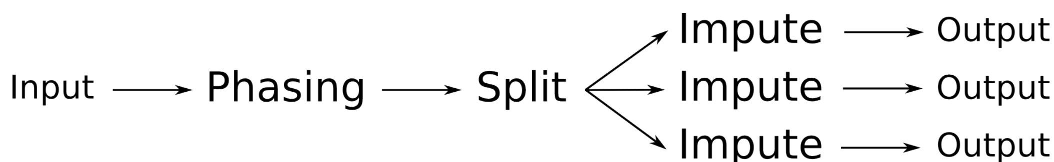

Figure 1 – A simplified sketch of the Imputation workflow for one chromosome. 22 of these are run in parallel. The number of Impute jobs depends on the chromosome, ranging from 11 to 62. For more details, see Fig. 9.

The *Imputation with pre-phasing* (later just *Imputation*) workflow consists of three consecutive stages: *phasing*, *splitting* and *imputation*. Each chromosome is considered separately, so the workflow has 22 parallel sub-workflows. The chromosomes have different sizes, leading to different CPU time usages and file sizes. The *splitting* stage splits the input into file into 11-62 outputs (depending on the size of the chromosome), for which the *imputation* step can

be done in parallel. A simplified sketch of the workflow for one chromosome is depicted in Fig. 1 and a more detailed description of the whole workflow is in Appendix A. In total, the workflow consists of 838 jobs and the total input file size is close to 450 GB. Total execution time is approximately 17 000 h on a single core, so a single 4-core computer would process the workflow for almost 6 months.

3.1.2 BWA/GATK

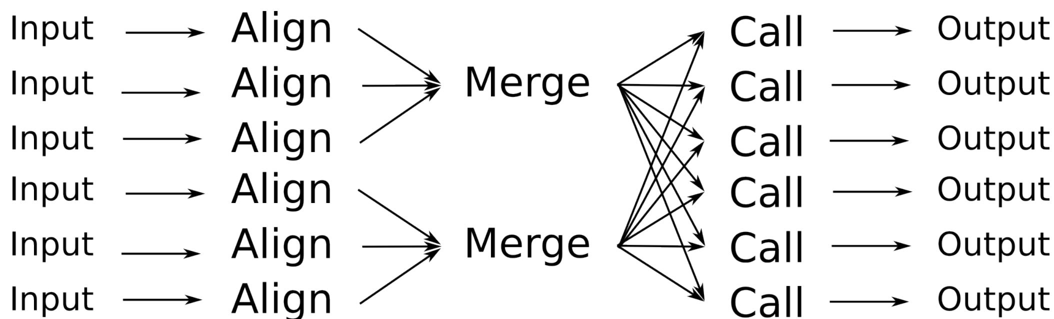

Figure 2 – A simplified sketch of the BWA/GATK workflow. For each sample, 16 Align jobs and 1 Merge job are executed. 10 samples form a callset, and for each callset the workflow contains 22 Call jobs (one per chromosome). For more details, see Fig. 10.

For the *BWA/GATK* workflow (Van der Auwera et al. [2002]), the input data is short genome reads from random locations in the genome. On average, the data covers the whole genome 10 times (the coverage varies for each part of genome due to the reads being from random locations), so most of the data is redundant. As the reads come from random locations, they must be aligned with respect to a reference genome before they can be used. Then a process know as *variant calling* can be used to extract the differences between the input genome and the reference genome. This results in a significant reduction of data, as the difference in the genomes between two (human) individuals is on the order of 0.1% (Jorde and Wooding [2004]).

Also the BWA/GATK example workflow consists of three consecutive stages: *alignment*, *merging* and *variant calling*. The *align*-stages (Li and Durbin [2009]) are run in parallel individually for each of the 16 input files per sample, after which the *merge* stage combines the aligned files and splits them by chromosome. The chromosome files are grouped into *callsets* of 10 samples. *Variant calling* is then done for each chromosome in each callset, so the stage consists of 22 jobs per 10 samples. Supposing that we have 1000

samples, this results in 19200 jobs, with input files totaling around 16 TB and total execution time being on the order of 178 000 h (over 20 years on one core). The workflow is shown in Fig. 2 and a more detailed description of the whole workflow is in Appendix B.

3.2 Example Environments

The example workflows will be scheduled to three different environments, details of which are provided by BC Platforms, representing typical calculation environments where BC Platform's software is deployed. To simplify the modeling of file transfers, each resource is assumed to have a disk of infinite capacity. The details for each resource are its CPU factor (assumed to be 1 unless stated otherwise), number of CPU cores, amount of memory (RAM) and the bandwidths to other resources. The actual execution time of a given job on a given resource depends only on the nominal execution time of the job divided by the CPU factor of the resource. The amount of memory and the number of CPU cores are only used to restrict the number of concurrent jobs a single resource can process.

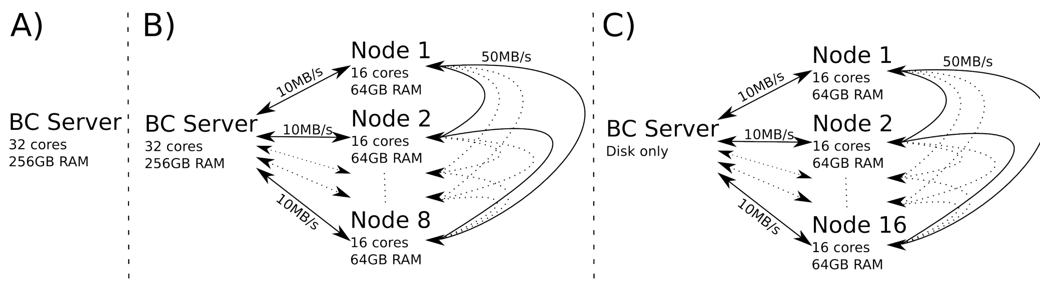

Figure 3 – The example environments. (A) is the single server environment. (B) includes additional 8 processing nodes with 16 cores and 64 GB of RAM. All the nodes are connected to the server with 10 MB/s links, and to each other with 50 MB/s links. For the third environment (C), the BC server is replaced by a disk, so that all calculations are done on the cloud nodes. Also the number of cloud nodes is doubled.

The first environment (set of available resources) (Fig. 3A) is a single server with 32 processors, 256 GB of memory and a 20 TB disk. This is a really simple environment corresponding to using just one server for the BC system. As there is only one resource, there are no transfer costs, and as the schedulers consider a calculation node as a single resource (cores are not considered separately), the only decision the schedulers have to make in the single-server environment is the order of jobs.

The B-environment is an expansion of the first one: in addition to the main server, it includes 8 cloud nodes with 16 processors, 64 GB memory and a 1 TB disk on each node. The input files are assumed to be on the server, and the output files should be there after the workflow has been executed. The data transfer speed between the cloud nodes is 50 MB/s and between the server and each node 10 MB/s. This corresponds to using a local server in addition to nodes from for example Amazon cloud. It is depicted in Figure 3B.

The C-environment is like the second, but with the local server replaced by just a disk, and the number of cloud nodes doubled. This corresponds to using calculation nodes from for example Amazon cloud with input and output files also being stored in the cloud (for example at an Amazon S3 disk). The disk-only resource is modeled with a really small CPU factor.

A randomized, heterogeneous version is created of the environment C by multiplying the CPU factors of each node as well as all the transfer speeds between disks by a random numbers from uniform distribution $[0.5, 1.5]$. The number of nodes, the numbers of cores and the amounts of memory are kept the same.

4 Results

In this section, the results (scheduling times and makespans) from the experiments described previously are presented and analyzed. Sections 4.1 4.2 focus on the Imputation and BWA/GATK workflows, respectively. The effect of heterogeneous resources is studied in Section 4.3, and the performance of the metaheuristics is analyzed in Section 4.4.

The results shown here are of from a single scheduling run. The Genetic Algorithm and GRASP are random in nature, but as both algorithms generate multiple random schedules (GRASP generates one per each iteration and GA the initial population), the results are very similar from one run to another. For Myopic, randomness may arise from the order on which jobs are scheduled, while with the other schedulers, the order of the tasks that are equal based on the criteria used by the scheduler may vary, depending on the implementation. The implementations used for this work are deterministic, so they produce the same schedule every time.

4.1 Imputation workflow

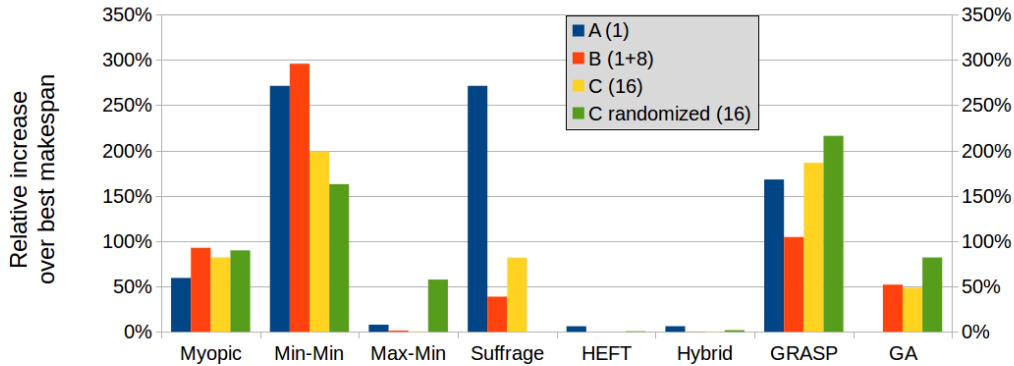

Figure 4 – The relative increases of the makespans of the schedules produced by each algorithm for the Imputation workflow on all three environments as well as the heterogeneous version of the environment C. The increases are calculated with relatively to the best schedule produced for each environment. The algorithms producing the best schedules are GA, HEFT, HEFT and Suffrage, respectively. The makespans were confirmed by simulating the workflow using the produced schedules. Differences between the schedule makespans and simulated makespans were negligible. GA was used with 100 generations and a population size of 20 schedules, GRASP used 30 iterations. The values of the makespans are listed in Tables 3-6.

Figure 4 shows a comparison of the makespans produced by each scheduler for the Imputation workflow on four different environments. The first thing to note here is that the makespans have a lot of variation, the longest taking almost four times as long as the shortest for the same environment. HEFT and Hybrid both produce schedules that are close to the shortest one on each environment. The Genetic Algorithm manages to produce the best schedule for the single-server case, but for other cases its makespans are around 50% longer than the shortest. Suffrage produces the best makespan for the heterogeneous environment, but for the single-resource case it produces the worst makespan together with Min-Min. The worst schedules are actually identical, as in a single-resource environment the Suffrage scheduler picks jobs in the same order as the Min-Min scheduler, due to its implementation. Taking into account the fact that a single resource might be able to execute multiple jobs in parallel would probably improve Suffrage's performance, but it is not trivial to do.

For this workflow, Min-Min produces the worst schedule on the three homogeneous environments, while the schedules given by Max-Min on the same

environments are really good. The reason is that the Imputation-workflow consists of tasks of different length (notably in the Phasing-step), and the longer tasks have much higher memory requirements than the shorter tasks, so that not many of those can be run at the same time on one resource. Thus, as the Max-Min schedules the longest tasks first, it can then fill the resources with smaller tasks and achieve higher parallelism. Min-Min, on the other hand, begins by scheduling the short tasks and is then left with just big tasks that take long time to execute and that cannot be run in parallel, leaving some of the cores idle.

The good performance of the Genetic Algorithm on the single-server environment is probably due to the height-ordering criteria, which forces each schedule to first have all the Phasing-jobs, then all Split-jobs and finally all Imputation-jobs. Thus the expensive Phasing-jobs are scheduled first.

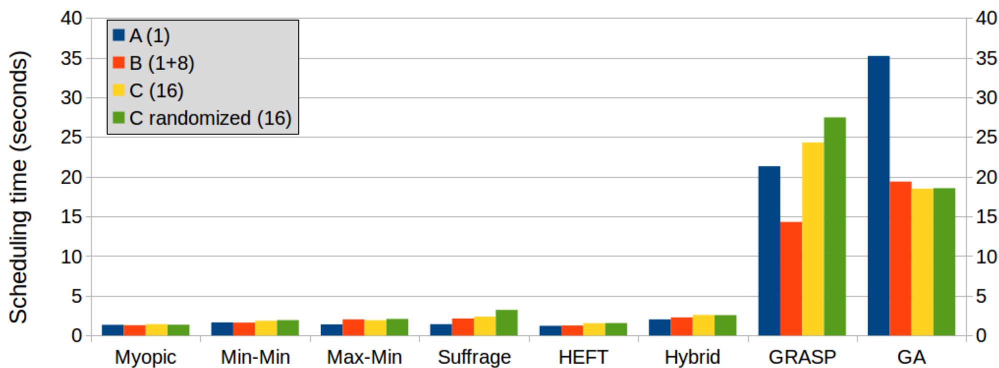

Figure 5 – The scheduling times for the Imputation workflow on all three environments as well as the heterogeneous version of the environment C. The scheduling times are listed in Tables 3-6.

Figure 5 shows the scheduling times for each algorithm and each execution environment, on the Imputation workflow. The heuristic algorithms are quite fast, with each running under 5 seconds, while the two metaheuristics take much longer ($15 - 35$ seconds). GA and GRASP use as much time as they are allowed to (by the number of generations or iterations, or other stopping criteria), so the results listed here are only meaningful with respect to the schedules they produce.

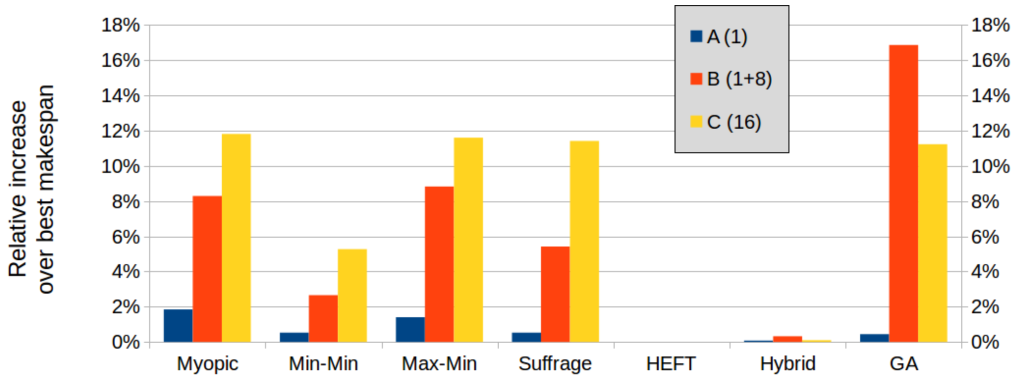

Figure 6 – The relative increases of the makespans of the schedules produced by each algorithm for the BWA/GATK workflow on all three environments A,B and C. The increases are calculated relatively to the best schedule produced for each environment - the one produced by HEFT, in each case. GRASP was not able to produce a schedule in reasonable time for any of the cases. GA was used with 100 generations and a population size of 20 schedules. The values of the makespans are listed in Tables 8-10.

4.2 BWA/GATK workflow

The makespans for the BWA/GATK workflow are shown in Fig. 6. Note that the BWA/GATK was not tested on the heterogeneous version of the C-environment, so there are just three values for each scheduler, corresponding to environments A,B and C. Also note that the GRASP scheduler has been left out as it was not able to produce a schedule in reasonable time (probably due to implementation-specific reasons).

For BWA/GATK, the relative differences between different makespans are much smaller than for Imputation, ranging from 1% to 17%. Still, the differences are not negligible: even for the single-resource environment the difference between the shortest and longest makespans is over 4 days, and for the B-environment it is over 8 days (Tbl. 8, 9).

The best schedules are produced by the HEFT scheduler, with Hybrid being very slightly behind. The baseline for the makespan on the single-resource environment is 231.7 days (178 000 h divided by 32 cores), so HEFT's result of 232.1 days (Tbl. 8) is really good. For the other schedulers, the relative increase grows with the number of resources, except for the Genetic Algorithm which performs worst on the slightly-heterogeneous environment B. The differences between Myopic, Max-Min and Suffrage are small, but Min-Min performs significantly better. This is in contrast with the results attained

with the Imputation-workflow, where Min-Min performed really badly compared to Max-Min.

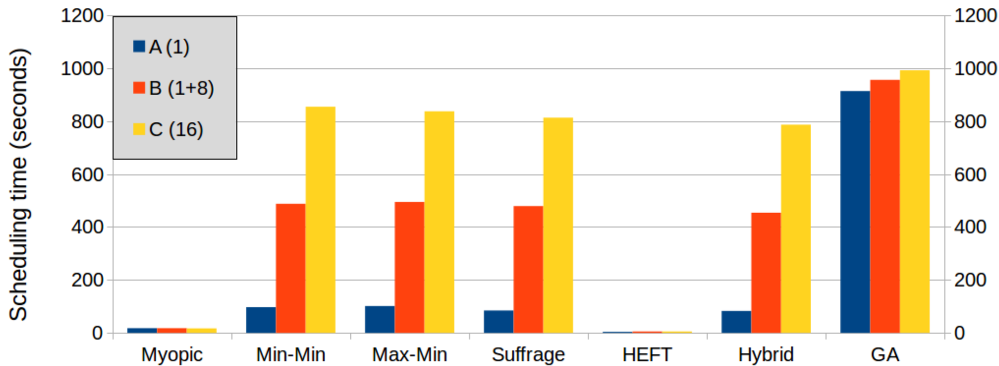

Figure 7 – The scheduling times for the BWA/GATK workflow on all three environments. GRASP was not able to produce a schedule in reasonable time for any of the cases. GA was used with 100 generations and a population size of 20 schedules. The scheduling times are listed in Tables 8-10.

Figure 7 shows the scheduling times for the BWA/GATK -workflow. Again HEFT and Myopic are both really fast and the GA has quite a long running time regardless of the set of available resources. Min-Min, Max-Min, Suffrage and Hybrid have all similar running times that seem to be related to the number of resources, which is in accordance with their presumed computational complexity (Yu et al. [2008]).

4.3 Heterogeneous resources

Figure 8 shows the importance of knowledge of the performance of the resources. First, simulating a schedule where resources have been assumed to be homogeneous on heterogeneous resources results in an significant increase in the makespan for Max-Min, HEFT and Hybrid schedulers – the schedulers that found the best schedules for the homogeneous environment – while the others are not that affected. Worth noting is that the simulator used does not even try to execute a job before the time given in the schedule. Second, on the heterogeneous environment, the schedules that are produced for it are significantly better than those produced for the homogeneous resources. This is due to the schedulers being able to better exploit the faster resources and avoid the slower ones. The biggest improvement is in the schedule by Suffrage (from 19 days for the simulated makespan to 8 days), but also the schedules produced by HEFT and Hybrid drop in half. Interesting to note

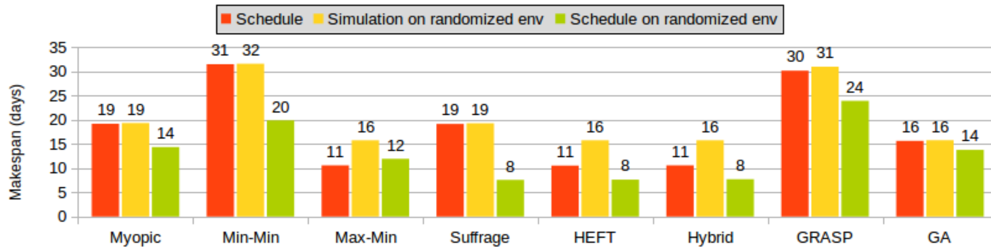

Figure 8 – The scheduled makespans for homogeneous environment C, makespans from simulation on the heterogeneous version and makespans of schedules made for the heterogeneous version, on the Imputation-workflow. The makespans of the two different schedules are not directly comparable, as the total processing power of the heterogeneous environment may differ from the processing power of the homogeneous version.

is that while the makespans for HEFT and Suffrage schedules are shorter than their homogeneous versions, Max-Min scheduler is not able to make use of the heterogeneous resources as well and its makespan is longer than the homogeneous one (while being still shorter than the homogeneous one when simulated on heterogeneous resources).

Another case would be one in which the exact execution times of the jobs are not known. It was not studied in this work, but it could be modeled in a similar way than unknown resources: produce a schedule using estimates of the execution times and evaluate the schedule with the simulator using randomized execution times. The results would probably also be similar, i.e. a significantly better schedule can be made if the estimates of the execution times are correct.

4.4 The Genetic Algorithm and GRASP

For the metaheuristics Genetic Algorithm and GRASP, in addition to the final makespan, also the efficiency of their iterations is of interest. If, for example, the genetic algorithm's makespan would improve significantly on each generation, it might be worth running for a longer time. However, from Table 1 we see that this is not the case. Thus, it is unlikely that increasing the number of generations would have any other significant effect than increasing the scheduling time. Also, the good performance of the GA scheduler on the Imputation workflow on single resource (Fig. 4) can be attributed to the height ordering criteria, which forces each schedule to begin by executing the Phasing-jobs (on any single resource the jobs must be

executed in ascending order of height). In most cases, however, the height ordering probably prevented the GA from reaching a good solution.

Workflow	Imputation				BWA/GATK		
Environment	A	B	C	C rand	A	B	C
First generation	2176061	1419487	1364774	1197430	2849340	4755335	2883810
Last generation	2159798	1391166	1350046	1188484	2822835	4735594	2846408
Improvement	0.75%	2.04%	1.09%	0.75%	0.94%	0.42%	1.31%

Table 1 – The makespans of the best schedule in the first generation and in the last generation (both in seconds) and their difference (relative to the final schedule). The improvements are really small, suggesting that the genetic operations do not work well on these cases.

For GRASP, the effect of the local optimization is harder to quantize, but it seemed that for all test cases the local optimization was able to improve at least some of the candidate schedules, and in most cases the final schedule was a result of the local search (Tbl. 2). As GRASP is a random scheduler, the results might vary from one run to another, but still it is clear that the local search does improve the results.

Environment	A	B	C	C rand
Candidates improved	3	4	10	10
Total improving swaps	38	9	28	26
Final schedule improved	yes	yes	yes	no

Table 2 – The effects of the local search phase of the GRASP scheduler on the Imputation workflow and each environment. The first row shows the number of candidate schedules that were improved by the local search, out of the 30 candidates produced. Second row shows the total number of swaps done by the local search in total for all candidate schedules. Third row indicates if the best of the candidate schedules was improved by the local optimization.

5 Conclusions

The aim of this work was to investigate different workflow scheduling algorithms, implement some of them and compare them on two example workflows and three example environments (sets of resources). The workflow scheduling problem was formulated as an optimization problem, but it was

noted to be NP-complete and not feasible to be solved exactly. To find an approximate solution, eight algorithms ranging from a really simple heuristic to full-blown metaheuristics were studied and implemented: Myopic, Min-Min, Max-Min, Suffrage, HEFT, Hybrid, GRASP and a Genetic Algorithm.

The algorithms were tested on two workflows, Imputation and BWA/GATK, both of which are commonly used analysis workflows in genetics. They share some properties, but also differ in many ways, showing different aspects of the scheduler algorithms. For resources, three different sets were used: a single server, the main server plus calculation nodes and just calculation nodes. The three examples represent common calculation environments where Biocomputing Platform's products are deployed.

The results from the experiments show that the choice of scheduler really matters, with the difference between the makespans of the worst and the best schedules being at worst close to 300% and over 10% in most cases. Big differences were also noted in the schedulers' running times when the number of jobs and resources were large, with the fastest being in the order of seconds and slowest taking tens of minutes. Still, scheduling times were found to be small compared to the makespans of the workflows, and as such no scheduler is too slow to be usable, even if in a real grid environment it would probably have to be run quite often as the availability of the resources may change over time and new jobs may need to be scheduled.

Based on the results, the Heterogeneous-Earliest-Finish-Time (HEFT) algorithm seems to both be the fastest and produce the best schedules, on the average. Even in the cases where some other scheduler produced a shorter makespan, the difference to the schedule produced by HEFT was negligible, meaning that HEFT is very robust with respect to the workflow and the resources. The results of the Hybrid algorithm were almost as good as HEFT's, but it is computationally much more expensive. The other algorithms that produced good schedules for some case produced really bad schedules for some other case, an example being Min-Min which was clearly the worst scheduler for the Imputation-workflow but came third for BWA/GATK.

Many simplifying assumptions had to be done to make the implementation of the schedulers possible in the given time frame. Most of the assumptions are related to file handling, as optimizing the file storage and transfers would probably be worth its own thesis. Most likely these assumptions do not play a significant role in the results, but it would still be interesting to confirm the performance of the schedulers on a real-world grid environment with real workflows and jobs. In a case where perfect information about the jobs and resources is not possible to get, the schedule will never perform exactly as

assumed by the scheduler unless the scheduler adds some slack time between jobs to account for longer-than-expected executions. The effect and optimal amount of slack time could also be studied in the future.

In addition to producing best schedules, the HEFT algorithm is also fastest. The running times, however, are very implementation-specific. By optimizing the implementations, it would probably be possible to reduce the running times of all other heuristics to be of the same magnitude as HEFT. Still, the HEFT algorithm's advantage is that it is really easy to implement in an efficient way: out of all the algorithms here, least time was probably used on implementing and optimizing the HEFT and Myopic schedulers.

When it comes to metaheuristics, based on these results the GRASP algorithm is not suitable for workflow scheduling on multi-resource environment. The Genetic Algorithm would require much work to be both more effective and to produce better schedules, but there is room for improvement. For example, the height criteria now prevents the GA from reaching the optimal solutions in some cases, but not using the criteria would mean much longer running time for the algorithm itself unless some other measure could be found to enforce the validness of the schedules produced by the genetic operations.

When using a so called utility grid, the user must usually pay for used resources. It is often cheaper to use less-capable resources, so a user might want the scheduler to also consider the cost of operations, meaning that the scheduling should be performed with these Quality of Service -constraints in mind, instead of just minimizing the makespan. The most usual QoS-constraints are budget and time constraints: for a budget constraint, the user wants a schedule that minimizes the makespan while costs stay under a given budget, while a time constraint means that the scheduler should minimize costs while keeping the makespan under the given time. Genetic algorithms can give a solution to both constraints, and there are also heuristics-based methods (Backtracking, deadline distribution for DL-constraint, LOSS and GAIN for budget-constraint) (Yu et al. [2008], Sakellariou et al. [2007]). For HEFT, no QoS-extensions were found.

To conclude, after comparing the algorithms both in theory and in practice, it would seem that the Heterogeneous-Earliest-Finish-Time (HEFT) algorithm would be the best choice as a scheduler for the situations tested in this work. Further study could focus on implementing improvements for the HEFT algorithm and testing their effects, as well as investigating the impact of different ranking methods to the makespan.

References

S Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende. A grasp for job shop scheduling. In *Essays and surveys in metaheuristics*, pages 59–79. Springer, 2002.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759–767. IEEE, 2005.

T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

I. Foster and C. Kesselman. *The Grid 2: Blueprint for a new computing infrastructure.* Morgan Kaufmann, 2003.

E.S.H. Hou, N. Ansari, and Hong R. A genetic algorithm for multiprocessor scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 5(2): 113–120, Feb 1994.

B. Howie, J. Marchini, and M. Stephens. Genotype imputation with thousands of genomes. *G3: Genes, Genomes, Genetics*, 1(6):457–470, 2011.

B. Howie, C. Fuchsberger, M. Stephens, J. Marchini, and G.R. Abecasis. Fast and accurate genotype imputation in genome-wide association studies through pre-phasing. *Nature genetics*, 44(8):955–959, 2012.

L.B. Jorde and S.P Wooding. Genetic variation, classification and'race'. *Nature genetics*, 36:S28–S33, 2004.

H. Li and R. Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

R. Sakellariou, H. Zhao, E. Tsiakkouri, and M.D. Dikaiakos. Scheduling workflows with budget constraints. In *Integrated Research in GRID Computing*, pages 189–202. Springer, 2007.

H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

J. D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences, vol. 10*, pages 384–393, 1975.

G.A. Van der Auwera, M.O. Carneiro, C. Hartl, R. Poplin, G. del Angel, A. Levy-Moonshine, T. Jordan, K. Shakir, D. Roazen, J. Thibault, E. Banks, K.V. Garimella, D. Altshuler, S. Gabriel, and M.A. DePristo. *From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline*. John Wiley & Sons, Inc., 2002.

M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *ACM SIGMOD Record*, 34(3):56–62, 2005.

J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *Sigmod Record*, 34(3):44, 2005.

J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. In *Metaheuristics for scheduling in distributed computing environments*, pages 173–214. Springer, 2008.

H. Zhao and R. Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Euro-Par 2003 Parallel Processing*, pages 189–194. Springer, 2003.

Appendices

A Imputation Workflow

The Imputation workflow, as it was given by BC Platforms and used in this work. Assumes whole genome imputation of 10000 subjects genotyped using a chip with $2.5M$ markers, imputed using $1000G$ imputation reference panels.

1. Phasing: 22 jobs, one per chromosome

   Input A study genotype files (one for each chromosome) with 10000 subjects. File size corresponds to chromosome size, in gigabytes: 34.80,35.34, 34.53, 29.86, 27.01, 30.79, 25.22, 23.09, 20.77, 21.48, 20.26, 22.44, 17.10, 15.90, 14.05, 15.65, 13.27, 13.56, 10.66, 11.01, 6.91, 6.30

   Output one phased genotype file per job, same size as input

   RAM corresponds to chromosome size, in gigabytes: 49.68, 50.46, 49.30, 42.63, 38.57, 43.96, 36.00, 32.97, 29.66, 30.67, 28.92, 32.05, 24.42, 22.70, 20.06, 22.35, 18.95, 19.37, 15.22, 15.71, 9.86, 8.99

   CPU 4 cores, execution time corresponds to chromosome size, in seconds: 850561, 863861, 844061, 729852, 660265, 752541, 616368, 564485, 507723, 525062, 495176, 548640, 418117, 388638, 343472, 382637, 324499, 331563, 260564, 269040, 168876, 153998

2. Splitting: 22 jobs, one per output of Phasing

   Input A phased genotype file (output of Phasing)

   Output The input splitted into chunks of 568 MB, resulting in a different number of files, corresponding to chromosome: 61, 62, 61, 53, 48, 54, 44, 41, 37, 38, 36, 40, 30, 28, 25, 28, 23, 24, 19, 19, 12, 11

   RAM negligible

   CPU 1 core, execution time negligible

3. Imputation: 794 jobs, one per output of Splitting. $MARKERS$ models the variable marker density in the reference panel chunks and is distributes according to the normal distribution with a mean of 49993 and variance of 7253.

Input A chunk of a phased genotype file (output of Splitting) and the respective chunk of the reference panel file ($MARKERS/50000 * 3.9\,\text{MB}$).

Output Genotype file of size $MARKERS * 9\,\text{GB}$

RAM $MARKERS/50000 * 10\,\text{GB}$

CPU 1 core, execution time $MARKERS/50000 * 6\,\text{h}$

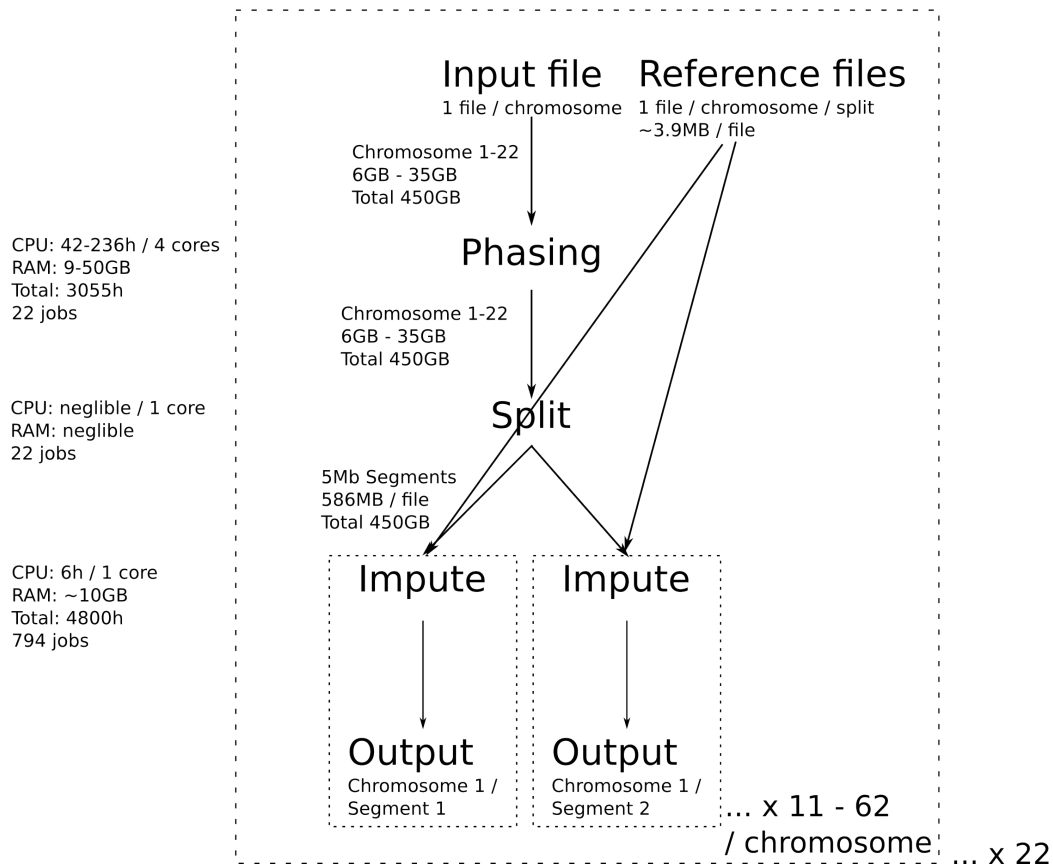

Figure 9 – Imputation workflow for one chromosome. 22 of these are run in parallel. Each parallel line has its own input file and results in $11 - 62$ output files. First operation is Phasing, after which the file is split into almost same-sized segments. Each segment is then imputed separately. Total number of jobs is 838, with total execution time around 17 000 h.

B BWA/GATK Workflow

The BWA/GATK workflow, as it was given by BC Platforms and used in this work. Assumes 1000 samples, 16 raw read files per sample and a callset of 10 samples.

1. Align: 16000 jobs, 16 per sample

   Input A raw read file (1 GB) and a reference sequence file (3 GB).

   Output Aligned read file (1 GB)

   RAM 4 GB

   CPU 1 core, execution time 3 h

2. Merge: 1000 jobs, 1 per sample

   Input Aligned read files for each sample (outputs of 16 Align-jobs)

   Output 22 files, one per chromosome, sizes corresponding to chromosomes (in gigabytes): 1.22, 1.24, 1.21, 1.05, 0.95, 1.08, 0.88, 0.81, 0.96, 0.75, 0.71, 0.79, 0.60, 0.56, 0.49, 0.55, 0.47, 0.48, 0.37, 0.39, 0.24, 0.22

   RAM 4 GB

   CPU 1 core, execution time 10 h

3. Variant Call: 2200 jobs, 22 per 10 samples

   Input A chromosome file of 10 samples

   Output VCF file, sizes corresponding to chromosomes (in megabytes): 53.36, 54.19, 52.95, 45.78, 41.42, 47.21, 38.66, 35.41, 41.83, 32.94, 31.06, 34.42, 26.23, 24.38, 21.55, 24.00, 20.36, 20.80, 16.34, 16.88, 10.59, 9.66

   RAM 4 GB

   CPU 1 core, execution time varies corresponding to chromosome (in hours): 91.47, 92.90, 90.77, 78.49, 71.00, 80.93, 66.28, 60.70, 71.70, 56.46, 53.25, 59.00, 44.96, 41.79, 36.94, 41.15, 34.90, 35.65, 28.02, 28.93, 18.16, 16.56

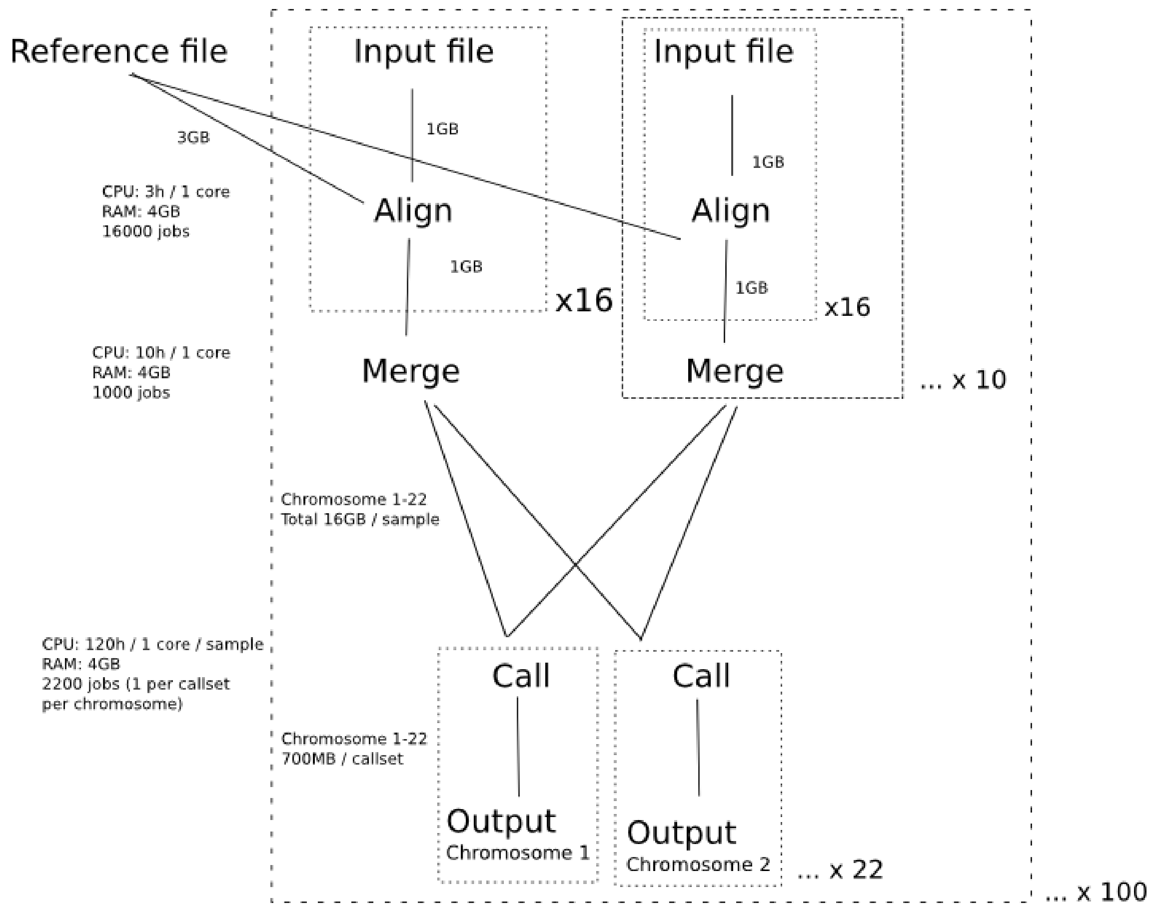

Figure 10 – The BWA/GATK workflow consists of 19200 jobs in three phases: Align, Merge and Call. As input files it has 1000 samples and 16 raw read files for each sample. Each input file is first aligned separately according to a reference file, and then the files for each single sample are merged and divided to chromosomes. Finally, for each chromosome in each callset (a set of 10 samples), variant calling is done. This gives in total 19200 jobs (16000 Align, 1000 Merge, 2200 Call), totaling at around 20 year's worth of CPU time.

C Results - Imputation

The results of scheduling the Imputation workflow for each environment are listed below. The *scheduling time* is as measured by the *Unix* program "time", as the sum of *user* and *system* times (CPU times taken by the process in user mode and kernel, respectively). *Makespan* is the time when the last job finishes. The *simulation*-column shows the simulated makespan, and the *difference between simulation and makespan* is the simulated makespan minus scheduled makespan (simulated makespan is always at least as long as the scheduled makespan).

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule	Simulation (days)	Difference (days)
Myopic	1	3443542	39.86	59.44%	39.86	0.00
Min-Min	2	8019889	92.82	271.33%	92.82	0.00
Max-Min	1	2330022	26.97	7.88%	26.97	0.00
Suffrage	1	8019889	92.82	271.33%	92.82	0.00
HEFT	1	2293101	26.54	6.17%	26.54	0.00
Hybrid	2	2293451	26.54	6.19%	26.54	0.00
GRASP	21	5789468	67.01	168.06%	67.01	0.00
GA	35	2159798	25.00	0.00%	25.00	0.00

Table 3 – The results of scheduling the Imputation workflow on the environment A. The best schedule is produced by the genetic algorithm.

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule	Simulation (days)	Difference (days)
Myopic	1	1763581	20.41	92.62%	20.41	0.00
Min-Min	2	3622948	41.93	295.69%	41.94	0.01
Max-Min	2	927523	10.74	1.30%	10.77	0.03
Suffrage	2	1270555	14.71	38.77%	14.71	0.01
HEFT	1	915592	10.60	0.00%	10.62	0.02
Hybrid	2	917716	10.62	0.23%	10.65	0.03
GRASP	14	1872757	21.68	104.54%	21.68	0.00
GA	19	1391166	16.10	51.94%	16.10	0.00

Table 4 – The results of scheduling the Imputation workflow on the environment B. The best schedule is produced by HEFT.

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule	Simulation (days)	Difference (days)
Myopic	1	1655173	19.16	82.06%	19.16	0.00
Min-Min	2	2717888	31.46	198.96%	31.47	0.01
Max-Min	2	912080	10.56	0.33%	10.61	0.06
Suffrage	2	1653114	19.13	81.84%	19.13	0.00
HEFT	2	909123	10.52	0.00%	10.54	0.02
Hybrid	3	912072	10.56	0.32%	10.57	0.02
GRASP	24	2603902	30.14	186.42%	30.14	0.00
GA	18	1350046	15.63	48.50%	15.63	0.00

Table 5 – The results of scheduling the Imputation workflow on the environment C. The best schedule is produced by HEFT.

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule	Simulation (days)	Difference (days)
Myopic	1	1239591	14.35	89.83%	14.35	0.00
Min-Min	2	1716462	19.87	162.86%	19.90	0.03
Max-Min	2	1029247	11.91	57.62%	12.02	0.10
Suffrage	3	652995	7.56	0.00%	7.61	0.05
HEFT	2	657257	7.61	0.65%	7.65	0.04
Hybrid	3	664739	7.69	1.80%	7.72	0.03
GRASP	27	2064691	23.90	216.19%	23.90	0.00
GA	19	1188484	13.76	82.01%	13.80	0.04

Table 6 – The results of scheduling the Imputation workflow on the heterogeneous version of environment C. The best schedule is produced by Suffrage.

	Makespan (days)	Simulation (days)	Simulation on randomized environment (days)	Makespan on randomized environment (days)
Myopic	14.35	14.35	19.32	14.35
Min-Min	19.87	19.90	31.57	19.87
Max-Min	11.91	12.02	15.74	11.91
Suffrage	7.56	7.61	19.27	7.56
HEFT	7.61	7.65	15.74	7.61
Hybrid	7.69	7.72	15.74	7.69
GRASP	23.90	23.90	30.95	23.90
GA	13.76	13.80	15.75	13.76

Table 7 – The difference between scheduling under performance guidance and without it, when scheduling the Imputation workflow on environment C. The "makespan" is the makespan of the schedule for the homogeneous version of the environment, and "simulation" is the simulated makespan. The "simulation on randomized environment" is the same schedule simulated on the heterogeneous version of the environment, and "makespan on randomized environment" is the makespan of the schedule generated for the heterogeneous version.

D Results - BWA/GATK

The results of scheduling the BWA/GATK workflow for each environment are listed below. The *scheduling time* is as measured by the *Unix* program "time", as the sum of *user* and *system* times (CPU times taken by the process in user mode and kernel, respectively). *Makespan* is the time when the last job finishes.

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule
Myopic	18	20421623	236.36	1.84%
Min-Min	97	20157354	233.30	0.53%
Max-Min	101	20332891	235.33	1.40%
Suffrage	85	20157354	233.30	0.53%
HEFT	4	20051760	232.08	0.00%
Hybrid	83	20067195	232.26	0.08%
GA	913	20140075	233.10	0.44%

Table 8 – The results of scheduling the BWA/GATK workflow on the environment A. The best schedule is produced by HEFT.

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule
Myopic	18	4369401	50.57	8.28%
Min-Min	487	4142365	47.94	2.66%
Max-Min	494	4391147	50.82	8.82%
Suffrage	479	4253780	49.23	5.42%
HEFT	5	4035209	46.70	0.00%
Hybrid	454	4048207	46.85	0.32%
GA	956	4714934	54.57	16.84%

Table 9 – The results of scheduling the BWA/GATK workflow on the environment B. The best schedule is produced by HEFT.

	Scheduling time (s)	Makespan (s)	Makespan (days)	Relative increase over best schedule
Myopic	17	2837565	32.84	11.80%
Min-Min	854	2671762	30.92	5.27%
Max-Min	837	2832215	32.78	11.59%
Suffrage	813	2827545	32.73	11.41%
HEFT	5	2538067	29.38	0.00%
Hybrid	786	2540586	29.40	0.10%
GA	992	2822835	32.67	11.22%

Table 10 – The results of scheduling the BWA/GATK workflow on the environment C. The best schedule is produced by HEFT.

E Yhteenveto (in Finnish)

Yksi nykyaikaisen tieteellisen tutkimuksen ominaispiirteitä on erittäin suuret tietomäärät, joiden analysointi vaatii tehokkaimmaltakin tietokoneelta paljon aikaa: tunteja, päiviä, viikkoja tai jopa kuukausia. Yhdeksi ratkaisuksi on esitetty analyysien jakamista pienempiin osiin (töihin) ja näiden osien suorittamista yhtäaikaisesti usealla tietokoneella (resurssilla). Samalla Internetin kehitys on mahdollistanut eri puolilla maailmaa sijaitsevien erilaisten resurssien liittämisen yhdeksi kokonaisuudeksi, jolla useat tutkijat voivat suorittaa analyysejansa. Jotta töiden suorittaminen yhtäaikaisesti onnistuisi järkevästi ja nopeasti, työt täytyy jakaa käytössä olevien resurssien kesken ja niiden suoritusjärjestys täytyy määrätä. Tätä kutsutaan aikataulutukseksi.

Kandidaatintyössä tutkittiin eri menetelmiä aikataulutuksen tekemiseen työnantajan, Biocomputing Platforms Ltd:n, antamassa viitekehyksessä. Esimerkkianalyyseinä käytettiin kahta genetiikan menetelmää, joista toisen tarkoituksena on täydentää puuttuvia mittaustuloksia perustuen tilastolliseen mallintamiseen ja toisen löytää poikkeavuuksia vertailukohtana olevaan tietokantaan. Kumpikin analyysi voidaan jakaa osiin, jotka voidaan suorittaa rinnakkain ja peräkkäin; tätä jakoa kutsutaan työnkuluksi. Työnkulku voidaan esittää suunnattuna asyklisenä graafina, jonka viivat kuvaavat töiden riippuvuussuhteita. Ensimmäisen menetelmän työnkulku koostuu noin 800 työstä jaettuna 22 rinnakkaiseen linjaan. Toinen työnkulku sisältää 19 200 työtä sadassa rinnakkaisessa linjassa. Esimerkkiympäristöinä, joille työnkulkuja aikataulutettiin, käytettiin seuraavia malleja: ”yksi tehokas tietokone”, ”yksi tehokas tietokone + 8 samanlaista, pienempitehoista tietokonetta”, ”16 samanlaista tietokonetta” ja ”16 erilaista tietokonetta”.

Tavoitteena oli löytää menetelmä, jonka tuottamien aikataulujen kokonaispituus on mahdollisimman lyhyt, mutta myös aikataulun luomiseen kestävä aika huomioitiin. Tehtävän mukaisten laskentaympäristöjen tila voi muuttua kesken työnkulun suorituksen, esimerkiksi jos jokin tietokoneista joudutaan sulkemaan tai verkkoyhteys johonkin koneeseen katkeaa. Tällöin aikataulutus täytyy tehdä uudestaan.

Aikataulutus on NP-täydellinen ongelma, eli sen ratkaisemiseen tunnetaan vain eksponentiaalisen ajan vaativia algoritmeja. Menetelmän tulisi kuitenkin toimia nopeasti suurillakin työmäärillä, eikä eksponentiaalinen algoritmi siten ole käytännössä mahdollinen ratkaisu. Tarvitaan siis menetelmiä, jotka löytävät riittävän hyvän ratkaisun lyhyessä ajassa, eli niin kutsuttuja heuristiikkoja.

Työtä varten testattavaksi valitut menetelmät toteutettiin käyttäen Java-ohjelmointikieltä. Toteutuksen helpottamiseksi tiettyjä asioita jätettiin huomioimatta, esimerkiksi useiden yhtäaikaisten tiedonsiirtojen vaikutusta tiedonsiirtonopeuteen ei huomioitu ja tietokoneiden tallennuskapasiteetit oletettiin äärettömiksi. Yksittäisen työn kestoon oletettiin vaikuttavan ainoastaan tietokoneen

suorituskykykertoimen, muiden resurssivaatimuksien rajoittaessa yhtäaikaisten töiden määrää. Lisäksi päätettiin keskittyä ainoastaan aikataulun minimoimiseen. Monimutkaisempia tavoitteita olisivat esimerkiksi tietokoneiden ja tiedonsiirtoväylien käytöstä syntyvien kustannusten minimoiminen siten, että aikataulun pituus on rajoitettu, tai pituuden minimointi siten, että kustannuksilla on yläraja. Näitä rajoitteita kutsutaan palvelunlaatukriteereiksi.

Työssä toteutettiin ja testattiin kahdeksan eri menetelmää, jotka kaikki ovat tunnettuja aikataulutukseen käytettyjä algoritmeja. Näistä osa on heuristiikkoja, jotka pyrkivät rakentamaan mahdollisimman hyvän ratkaisun jonkin ennalta määrätyn säännön mukaan, ja osa metaheuristiikkoja, jotka etsivät mahdollisimman hyvää ratkaisua kokeilemalla erilaisia vaihtoehtoja. Jokainen heuristiikka käy läpi työt valiten yhden kerrallaan ja asettaa sen suoritettavaksi tietokoneelle, jolla se valmistuu nopeimmin. Erona menetelmissä on se, miten ne valitsevat työt: Myopic valitsee satunnaisen työn, Min-Min lyhimmän, Max-Min pisimmän ja Suffrage sen, joka kärsisi eniten jäädessään valitsematta sillä kertaa. Hieman monimutkaisempia heuristiikkoja ovat HEFT ja Hybrid, jotka ensiksi järjestävät työt tärkeysjärjestykseen niiden vaatimuksien sekä niitä seuraavien töiden mukaan ja sitten aikatauluttavat ne tämän järjestyksen mukaisesti. Tapoja tärkeyksien laskemiseen on useita, ja eri tapojen on havaittu toimivan hyvin eri tilanteissa. Metaheuristiikkoja toteutettiin kaksi: GRASP ja geneettinen algoritmi. GRASP luo satunnaisia ratkaisuja, pyrkii parantamaan ratkaisuja esimerkiksi vaihtamalla peräkkäisten töiden paikkoja ja valitsee lopulta parhaan. Geneettinen algoritmi pohjautuu nimensä mukaan genetiikkaan ja pyrkii löytämään ratkaisun käyttämällä evoluutiobiologian keinoja, jotka ovat rekombinaatio, mutaatio ja valinta.

Koska kaikki algoritmit ohjelmoitiin kandityötä varten, niihin tulee suhtautua sen mukaisella kriittisyydellä. Metaheuristiikat ovat toteutukseltaan huomattavasti heuristiikkoja monimutkaisempia, ja erilainen toteutus olisi voinut antaa erilaisia lopputuloksia. Erityisesti menetelmien käyttämät ajat ovat hyvinkin toteutuksesta määräytyviä. Osaa menetelmistä voi säätää erilaisilla asetuksilla, jotka vaikuttavat sekä aikataulun pituuteen että sen luomisen viemään aikaan. Työssä käytetyt parametrit on säädetty testauksen aikaisten kokemuksien mukaan sopiviksi.

Testissä huomattiin, että aikataulutusmenetelmän valinnalla voi olla todella suuri merkitys aikataulun pituuden kannalta: pahimmillaan pisin aikataulu oli neljä kertaa lyhimmän pituinen, ja useimmissa tapauksissa eroa oli vähintään 10 %. Menetelmien toimivuuteen vaikutti sekä työnkulku että ympäristö, ja useimmat menetelmistä toimivat hyvin ainakin yhdessä tilanteessa. Poikkeuksena Myopic ja GRASP loivat melko pitkiä aikatauluja kaikissa testitapauksissa, mistä voidaan todeta Myopicin olevan liian yksinkertainen ja GRASPin sopivan huonosti tähän tehtävään.

HEFT-algoritmi puolestaan tuotti jokaisessa testissä joko parhaan tai hyvin lähelle parasta olevan aikataulun. Lisäksi se oli nopein yhdessä Myopic-heuristiikan kanssa, ja tuotti aikataulun kaikissa tapauksissa alle 10 sekunnissa. Aikatauluttaessa jälkimmäistä, suurempaa työnkuvausta yhdelle tietokoneelle HEFT:n luoma aikataulu on lähellä teoreettista alarajaa, eli töiden yhteenlaskettua pituutta jaettuna tietokoneen prosessorien määrällä. Hybrid-menetelmän aikataulujen pituudet ovat jokaisessa testitapauksessa hyvin lähellä HEFTin aikataulujen pituuksia, mutta itse algoritmi vaatii paljon enemmän aikaa aikataulun tuottamiseen kuin HEFT.

Mielenkiintoinen pari vertailtavaksi on Min-Min ja Max-Min, joiden pieni ero johtaa hyvin erilaisiin lopputuloksiin. Kirjallisuuden perusteella Min-Min toimii yleensä hieman paremmin, mutta Max-Min voi antaa parempia tuloksia, mikäli työnkulku koostuu suuresta määrästä pieniä töitä ja muutamasta suuresta työstä. Tällaisessa tapauksessa Min-Min aikatauluttaa ensiksi pienet, lyhyet työt. Mikäli kaikkia suuria töitä ei voi suorittaa yhtä aikaa, rinnakkaisuus kärsii, sillä pieniäkään töitä ei ole enää jäljellä suurien töiden sekaan asetettavaksi. Tällainen tilanne on esimerkeistä ensimmäisessä työnkulussa, ja se näkyy tuloksissa: Max-Minin aikataulut ovat huomattavasti Min-Minin aikatauluja lyhyempiä. Min-Min puolestaan pärjää paremmin jälkimmäisessä työnkulussa, jossa töiden resurssivaatimukset eivät ole yhtä rajoittavia. Suffrage-algoritmi puolestaan vaikuttaa toimivan parhaiten silloin, kun käytössä olevat tietokoneet ovat erilaisia.

Geneettinen algoritmi (GA) onnistui tuottamaan parhaan aikataulun yhteen testitapauksista, jolloin ero toiseksi tulleeseen aikatauluun oli noin 6 %. GA:n voitto ei kuitenkaan ollut geneettisten operaatioiden ansiota, sillä ensimmäisen ja viimeisen sukupolven parhaiden aikataulun välillä oli vain noin 1 % ero, eli GA:n toimivuus kyseisessä tilanteessa johtui hyvästä alkupopulaatiosta. Geneettinen algoritmi laskee jokaiselle työlle ”korkeuden”, joka kuvastaa sen etäisyyttä työnkulun alusta. Geneettiset operaatiot käyttävät tätä korkeuslukua hyväkseen tuottaakseen aikatauluja, jotka ovat käypiä eli joissa töiden väliset riippuvuussuhteet toteutuvat. Esimerkiksi kahden samalla korkeudella olevan työn täytyy olla toisistaan riippumattomia, joten aikataulu pysyy käypänä, vaikka mutaatio vaihtaisi niiden paikat päittäin. Jokaisessa geneettisen algoritmin tuottamassa aikataulussa työt suoritetaan korkeusjärjestyksessä, ja käytännössä tämän järjestyksen pakottaminen johtaa joissain tapauksissa hyvään aikatauluun ja toisissa tapauksissa huonoon. Huomattavaa on, että geneettinen algoritmi voitaisiin toteuttaa myös ilman korkeusjärjestystä, jolloin lopputulokset saattaisivat olla parempia, mutta geneettisten operaatioiden toteutus tehokkaasti olisi huomattavasti haastavampaa.

Työssä todettiin, että tarkastelluissa, melko yksinkertaisissa tapauksissa, HEFT-algoritmi tuottaa nopeasti parhaan tai ainakin hyvin lähelle parasta olevan aikataulun. Seuraavaksi tulisi varmistua algoritmin toimivuudesta todellisessa ympäristössä ja todellisilla työnkuluilla sekä tutkia eri tärkeysjärjestyksen laskentatapojen vaikutusta aikatauluun. HEFTin perusversio ei kuitenkaan ole kykenevä huomioimaan palvelunlaatukriteerejä, toisin kuin esimerkiksi metaheuristiikat. Mikäli tulevaisuudessa halutaan ottaa huomioon muitakin kriteerejä kuin pelkkä aikataulun pituus, myös geneettisen algoritmin jatkokehitys voi olla hyödyllistä.