

Large-Scale Parallelization of Partial Evaluations in Evolutionary Algorithms for Real-World Problems

Anton Bouter
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
Anton.Bouter@cwi.nl

Tanja Alderliesten
Academic Medical Center
Amsterdam, The Netherlands
T.Alderliesten@amc.uva.nl

Arjan Bel
Academic Medical Center
Amsterdam, The Netherlands
A.Bel@amc.uva.nl

Cees Witteveen
Delft University of Technology
Delft, The Netherlands
C.Witteveen@tudelft.nl

Peter A.N. Bosman
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
Peter.Bosman@cwi.nl

ABSTRACT

The importance and potential of Gray-Box Optimization (GBO) with evolutionary algorithms is becoming increasingly clear lately, both for benchmark and real-world problems. We consider the GBO setting where partial evaluations are possible, meaning that sub-functions of the evaluation function are known and can be exploited to improve optimization efficiency. In this paper, we show that the efficiency of GBO can be greatly improved through large-scale parallelism, exploiting the fact that each evaluation function requires the calculation of a number of independent sub-functions. This is especially interesting for real-world problems where often the majority of the computational effort is spent on the evaluation function. Moreover, we show how the best parallelization technique largely depends on factors including the number of sub-functions and their required computation time, revealing that for different parts of the optimization the best parallelization technique should be selected based on these factors. As an illustration, we show how large-scale parallelization can be applied to optimization of high-dose-rate brachytherapy treatment plans for prostate cancer. We find that use of a modern Graphics Processing Unit (GPU) was the most efficient parallelization technique in all realistic scenarios, leading to substantial speed-ups up to a factor of 73.

CCS CONCEPTS

• **Mathematics of computing** → **Evolutionary algorithms**;

KEYWORDS

Gray-box optimization, parallel, CUDA, GPU, GOMEA

ACM Reference Format:

Anton Bouter, Tanja Alderliesten, Arjan Bel, Cees Witteveen, and Peter A.N. Bosman. 2018. Large-Scale Parallelization of Partial Evaluations in Evolutionary Algorithms for Real-World Problems. In *GECCO '18: Genetic*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO '18, July 15–19, 2018, Kyoto, Japan

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5618-3/18/07...\$15.00

<https://doi.org/10.1145/3205455.3205610>

and *Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan*.
ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3205455.3205610>

1 INTRODUCTION

Recent advances in evolutionary computation have shown that problems with millions of variables can be efficiently solved in a Gray-Box Optimization (GBO) setting, in the domains of both discrete [7] and real-valued optimization [4]. Optimization with Evolutionary Algorithms (EAs) in a GBO setting has also led to large improvements in optimization speed for various real-world problems, such as the traveling salesman problem [22], multi-objective deformable image registration [3], and the optimization of High-Dose-Rate (HDR) brachytherapy (BT) treatment plans for prostate cancer [17]. These problems have well-understood objective functions, but are nevertheless non-trivial to optimize. In the GBO setting that we consider, the objective functions are defined in a way that allows partial evaluations to be performed, meaning that the objective values of a solution can be efficiently updated after the modification of only a subset of variables.

The efficiency of an EA is frequently said to be easily improved through parallelization. The evaluation of all solutions in the population can be trivially parallelized, leading to large speed-ups for problems with a computationally expensive evaluation function. However, as the clock speed of Central Processing Units (CPUs) is starting to reach its physical limits [6], high-performance computing trends have recently focused on large-scale parallelism, for example using Graphics Processing Units (GPUs). Modern GPUs are equipped with several thousands of cores, but are restricted by the fact that they require a fine-grained parallelization model to be effective. In the field of evolutionary computation, GPUs are becoming widely used to speed-up various algorithms through the parallelization of their most time-consuming operations [9, 16, 19, 23].

In various real-world problems, evaluations require the majority of the computation time. Large-scale parallelization of the internal mechanisms of the EA will in that case only lead to marginal improvements when in a black-box setting. Instead, the parallelization of function evaluations could then lead to much larger improvements in performance. The degree of parallelization of evaluations is however limited by the population size, which rarely comes close to the thousands of cores that current-day GPUs are equipped with.

In a GBO setting, a much higher degree of parallelism can be achieved compared to a Black-Box Optimization (BBO) setting, because the objective function consists of known sub-functions that can be calculated independently. For each partial evaluation, a subset of these sub-functions has to be recalculated, which can be done in parallel. Because the population size, the number of functions to which partial evaluations can be applied, and the number of sub-functions in each decomposable evaluation function, all multiplicatively increase the possible degree of parallelism, large-scale parallelization quickly becomes feasible in GBO. We therefore apply large-scale parallelization to evaluation functions in a GBO setting, more specifically to partial evaluations, in order to greatly reduce the required computation time of partial evaluations.

In this paper, we introduce a general approach for the large-scale parallelization of partial evaluations in an EA in a GBO setting. We compare parallelization techniques using a GPU, an Intel Xeon Phi, and multiple Central Processing Unit (CPU) cores, and we analyze which kind of parallelization approach is preferred in a range of different scenarios, depending on factors including the computational effort of a partial evaluation and the number of sub-functions that are calculated in parallel. We then extend and validate this analysis with the real-world optimization problem of HDR BT treatment planning for prostate cancer.

2 GRAY-BOX OPTIMIZATION

In a GBO setting, some information on the optimization function is available. For example, a setting in which (part of) the definition of the optimization function is known can be considered a GBO setting if this information can be exploited to improve optimization.

One way of exploiting the GBO setting in (discrete) pseudo-Boolean optimization is by using partition crossover [21]. This approach does not easily generalize to real-valued optimization, as it works by finding the best among a finite set of possible offspring.

A more general way of exploiting GBO is by using partial evaluations. These are performed to efficiently update the objective value after the modification of a (small) subset of variables by recomputing the contribution of the modified variables to the objective value. Such partial evaluations were first discussed in [2] to be used with the discrete Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) [20], and were recently demonstrated with the Real-Valued GOMEA (RV-GOMEA) [4] to have the potential to substantially (by orders of magnitude) improve performance.

As a simple example of a problem to which partial evaluations can be applied, consider the well-known multi-objective benchmark problem ZDT1 [10], defined as

$$\begin{aligned} f_1^{\text{ZDT1}}(\mathbf{x}) &= \mathbf{x}_1, \\ f_2^{\text{ZDT1}}(\mathbf{x}) &= g(\mathbf{x}) \cdot h(f_1^{\text{ZDT1}}(\mathbf{x}), g(\mathbf{x})), \\ g(\mathbf{x}) &= 1 + \frac{9}{\ell - 1} \sum_{i=2}^{\ell} \mathbf{x}_i, \\ h(f(\mathbf{x}), g(\mathbf{x})) &= 1 - \sqrt{\frac{f(\mathbf{x})}{g(\mathbf{x})}}, \end{aligned}$$

with ℓ the number of variables, which are real-valued in ZDT1.

Consider a solution \mathbf{x}^g at time g , and the solution \mathbf{x}^{g+1} , which is the state of solution \mathbf{x} at time $g + 1$ after the modification of q variables. The $O(\ell)$ time complexity of the ZDT1 function is caused

by the calculation of $\sum_{i=2}^{\ell} \mathbf{x}_i^{g+1}$ required for the function $g(\mathbf{x}^{g+1})$. However, given the result of $\sum_{i=2}^{\ell} \mathbf{x}_i^g$, the result of $\sum_{i=2}^{\ell} \mathbf{x}_i^{g+1}$ can be computed in $O(q)$ time, because new values of the q modified variables can be added to the sum, and previous values can be subtracted. For this reason, we maintain the result of $\sum_{i=2}^{\ell} \mathbf{x}_i^g$ in memory, and update it after any modification of \mathbf{x} . The value of $g(\mathbf{x}^{g+1})$ can then be calculated in constant time, resulting in a total complexity of $O(q)$ if q variables are modified. In a similar way, partial evaluations can be applied to the other ZDT functions [10].

Applying partial evaluations to the ZDT functions requires only $O(n)$ memory for a population P of size n , but introduces the possibility of applying partial evaluations with a time complexity of $O(q)$, compared to full evaluations with a time complexity of $O(\ell)$.

2.1 General Representation

Given k sub-functions of $f(\mathbf{x})$, a function to which partial evaluations can be applied, let $\mathbf{I} = \{\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k\}$ where \mathcal{I}_j is the set of indices of problem variables on which sub-function f_j^M is dependent. Given a solution \mathbf{x} , a general representation of $f(\mathbf{x})$ is:

$$\begin{aligned} f(\mathbf{x}) &= f^P \left(f_1^M(\mathbf{x}|_{\mathcal{I}_1}) \oplus f_2^M(\mathbf{x}|_{\mathcal{I}_2}) \oplus \dots \oplus f_k^M(\mathbf{x}|_{\mathcal{I}_k}) \right) \\ &= f^P \left(\bigoplus_{j=1}^k f_j^M(\mathbf{x}|_{\mathcal{I}_j}) \right), \end{aligned}$$

where each $f_j^M(\mathbf{x}|_{\mathcal{I}_j})$ is a sub-function that depends on $\mathbf{x}|_{\mathcal{I}_j}$, which is a subset of \mathbf{x} restricted to the indices in $\mathcal{I}_j \subseteq \{1, \dots, \ell\}$.

To apply partial evaluations to $f(\mathbf{x})$, consider the state \mathbf{x}^g of solution \mathbf{x} at time g , and the state \mathbf{x}^{g+1} at time $g + 1$. In this example we assume that only the variable \mathbf{x}_i was modified, i.e., $\mathbf{x}_j^g = \mathbf{x}_j^{g+1}$ for $j \neq i$. If the operator \oplus has an inverse \ominus , then:

$$f(\mathbf{x}^{g+1}) = f^P \left(\bigoplus_{j=1}^k f_j^M(\mathbf{x}^g|_{\mathcal{I}_j}) \oplus \bigoplus_{\mathcal{I}_j \ni i} f_j^M(\mathbf{x}^{g+1}|_{\mathcal{I}_j}) \ominus \bigoplus_{\mathcal{I}_j \ni i} f_j^M(\mathbf{x}^g|_{\mathcal{I}_j}) \right),$$

with $\mathcal{I}_j \ni i$ shorthand for $\{\mathcal{I}_j \in \mathbf{I} \mid i \in \mathcal{I}_j\}$. For all dependent sub-functions, the contributions of \mathbf{x}^{g+1} are added with the \oplus operator, while the contributions of \mathbf{x}^g are subtracted with the \ominus operator. The calculation of $f(\mathbf{x}^{g+1})$ can be made more efficient by maintaining the sum of sub-functions, denoted s^g , in memory. The value of s^g is recursively updated as follows:

$$\begin{aligned} s^1 &= \bigoplus_{j=1}^k f_j^M(\mathbf{x}^1|_{\mathcal{I}_j}), \\ s^{g+1} &= s^g \oplus \left[\bigoplus_{\mathcal{I}_j \ni i} f_j^M(\mathbf{x}^{g+1}|_{\mathcal{I}_j}) \right] \ominus \left[\bigoplus_{\mathcal{I}_j \ni i} f_j^M(\mathbf{x}^g|_{\mathcal{I}_j}) \right]. \end{aligned}$$

For the modification of more than one variable, i.e., variables \mathbf{x}_i with indices $i \in \mathcal{F} \subseteq \{1, \dots, \ell\}$, s^g is updated as follows:

$$s^{g+1} = s^g \oplus \left[\bigoplus_{(\mathcal{F} \cap \mathcal{I}_j) \ni i} f_j^M(\mathbf{x}^{g+1}|_{\mathcal{I}_j}) \right] \ominus \left[\bigoplus_{(\mathcal{F} \cap \mathcal{I}_j) \ni i} f_j^M(\mathbf{x}^g|_{\mathcal{I}_j}) \right].$$

with $(\mathcal{F} \cap \mathcal{I}_j) \ni i$ shorthand for $\{\mathcal{I}_j \in \mathbf{I} \mid i \in (\mathcal{I}_j \cap \mathcal{F})\}$. After the recursive update of s^{g+1} , the partial evaluation of $f(\mathbf{x}^{g+1})$ is then calculated as:

$$f(\mathbf{x}^{g+1}) = f^P(s^{g+1}).$$

If the \oplus operator is the sum operator, then it trivially follows that \ominus is the minus operator. However, if \oplus is the multiplication operator, then using division for the \ominus operator leads to problems if the result of a sub-function is equal to zero. To overcome this problem, two values are maintained in memory instead of just s^{partial} : the product of all non-zero sub-functions $p^{\text{non-zero}}$, and the number of sub-functions equal to zero n^{zero} . The result of a sub-function (a, b) is then added to these values using the operators \oplus and \ominus in the following way:

$$(p^{\text{non-zero}}, n^{\text{zero}}) \oplus (a, b) = (a \cdot p^{\text{non-zero}}, n^{\text{zero}} + b),$$

$$(p^{\text{non-zero}}, n^{\text{zero}}) \ominus (a, b) = \left(\frac{p^{\text{non-zero}}}{a}, n^{\text{zero}} - b \right),$$

with each sub-function returning a tuple: $(1, 1)$ if its result is equal to zero, and $(f_j^M(\mathbf{x}^g|_{\mathcal{I}_j}), 0)$ otherwise. The final result of $f(\mathbf{x}^{g+1})$ is then equal to:

$$f(\mathbf{x}^{g+1}) = \begin{cases} f^P(0) & \text{if } n^{\text{zero}} > 0 \\ f^P(p^{\text{non-zero}}) & \text{if } n^{\text{zero}} = 0. \end{cases}$$

Partial evaluations can also be applied to more complex functions when the objective function can be written as a set of nested functions of which a sub-function decomposition is known. Moreover, the function f^P could require the calculation of multiple functions to which partial evaluations can be applied, as is the case in the optimization of HDR BT treatment plans, described in Section 6. These functions can also be calculated in parallel.

2.2 Gray-Box Optimization Algorithms

Partial evaluations have recently been applied to RV-GOMEA [4] and the Multi-Objective RV-GOMEA (MO-RV-GOMEA) [5], achieving excellent scalability on various benchmark problems [4] and real-world problems [3, 17] in a GBO setting. Partial evaluations are well-suited for use with algorithms of the GOMEA family, because the Gene-pool Optimal Mixing (GOM) variation operator applies variation to (small) subsets of variables, and reverts this change if it did not lead to an improved objective value. This selection step means that the change of a small subset of variables must be evaluated, which is done efficiently using partial evaluations.

Current state-of-the-art EAs for real-valued BBO [11, 13] apply variation to all variables of a solution, meaning that partial evaluations provide no benefit by themselves. However, when such algorithms are used in a GBO setting that allows partial evaluations, sub-functions of the evaluation function can still be calculated in parallel. This means that EAs aimed at BBO can still benefit from large-scale parallelization, as discussed in Section 4.

3 PARALLEL CO-PROCESSOR ARCHITECTURES

In this section we discuss two frequently used parallel co-processors and their benefits, as these architectures could be used for large-scale parallelization in GBO. We discuss the Intel Xeon Phi and the NVIDIA Pascal architectures, with the former aimed at coarse-grained parallelism, and the latter at fine-grained parallelism.

3.1 Intel Xeon Phi

An Intel Xeon Phi is a co-processor consisting of a large number of older, but general purpose, CPUs [14]. Some benefits of an Intel Xeon Phi are that it takes minimal effort to program applications for it, and that a coarse parallelization approach can be used. We specifically use an Intel Xeon Phi of the 5100 series with the Knight's Corner architecture. This co-processor has 60 cores at a base clock speed of 1.05 GHz each.

The Intel Xeon Phi co-processor can be used either in native, or in offload mode. In native mode, any program is executed on the co-processor similar to how it is executed on any multi-core computer. One of the cores of the co-processor executes the serial code, and the remaining cores of the co-processor are used to execute parallel regions of the code. A benefit of using the native mode is that there is no overhead caused by the copying of memory. Furthermore, an application designed for parallel execution on a multi-core computer can be directly executed on the co-processor in native mode.

In offload mode, the serial code of the program is executed on a core of the host machine, and parallel regions in the code can be offloaded to be executed on the co-processor. A benefit of the offload mode is that it can be combined with different parallel co-processors, such as a GPU, because the serial code is executed on the host machine. This comes at the cost of overhead caused by memory transfer between the host and the co-processor device.

3.2 NVIDIA Pascal

One of the most recent architectures used by NVIDIA GPUs, succeeding the Maxwell architecture, is the Pascal architecture. The Pascal architecture is used by the NVIDIA Titan X GPU, which we use in this paper to run experiments.

The NVIDIA Titan X has 12 GB of main device memory, which is completely separated from the main host memory. Any data that has to be accessed on the GPU needs to be copied from the host to the device memory.

The Pascal architecture consists of a large number of Compute Unified Device Architecture (CUDA) cores that perform computations, which are distributed among a number of Streaming Multiprocessors (SMs). Each SM contains a number of CUDA cores, schedulers, cache memory, and registers. Access of the shared memory of an SM is much faster than access of the global GPU memory, so frequently reused data should be stored in shared memory.

Different types of CUDA cores are used for single-precision or double-precision operations. Each SM of the NVIDIA Titan X is equipped with 128 CUDA cores for single-precision, and 4 CUDA cores for double-precision operations. Due to this 32:1 ratio, double-precision operations should be avoided on similar GPUs. The NVIDIA Titan X has 28 SMs, for a total of 3584 single-precision CUDA cores, each at a base clock speed of 1.4 GHz.

GPUs use the Single Instruction Multiple Data (SIMD) computation model. This means that each instruction is applied to a large number of data points in parallel. The CUDA [8] programming language, which we use to program on an NVIDIA GPU, applies the SIMD computation model by using *kernels*, which are functions that are executed by a grid of threads in parallel. Such a grid consists of a large number of blocks, and each block itself is a grid of threads. Each thread executes the same operations applied to

different data, depending on the coordinates of the thread and the block it is located in. A block should consist of a multiple of 32 threads, because each SM applies operations to sets of 32 threads, called *warps*, in parallel.

4 PARALLEL GRAY-BOX OPTIMIZATION

We distinguish three phases in the application of partial evaluations, because this allows us to identify sections to which a fine-grained parallelization approach can be applied. Given the modification of variable \mathbf{x}_i^g of solution \mathbf{x}^g , resulting in a new solution \mathbf{x}^{g+1} , these phases are described as follows:

- (1) *Map*
 - Compute $f_j^M(\mathbf{x}^g|_{I_j})$ and $f_j^M(\mathbf{x}^{g+1}|_{I_j})$ for all $I_j \ni i$ in parallel.
- (2) *Reduce*
 - $\Delta s = \bigoplus_{I_j \ni i} f_j^M(\mathbf{x}^{g+1}|_{I_j}) \ominus \bigoplus_{I_j \ni i} f_j^M(\mathbf{x}^g|_{I_j})$.
 - $s^{g+1} = s^g \oplus \Delta s$.
- (3) *Process*
 - $f(\mathbf{x}^{g+1}) = f^P(s^{g+1})$.

4.1 CPU

When a number of CPU cores much smaller than the population size is used, no more can be achieved than the straightforward parallel evaluation of a number of solutions in the population. Therefore, each thread simply performs each phase sequentially for a single solution in the population at a time.

When using the Intel Xeon Phi co-processor, partial evaluations are applied to different solutions in the population in parallel, and any available threads are used for the parallel calculation of sub-functions during the *map* phase.

4.2 GPU

On a GPU, each phase is applied in parallel to the entire population P^g , and synchronization is required between consecutive phases. The results of the *map* phase are saved in a $(k \times 2n)$ matrix \mathbf{B} , with k the number of calculated sub-functions. For each modified solution P_j^{g+1} the row $\mathbf{B}_{*,2j-1}$ is used to store the results of all sub-functions, and the results for all sub-functions of solution P_j^g are saved in the row $\mathbf{B}_{*,2j}$, as displayed in Figure 1. The CUDA Thrust library [1], which includes many CUDA utility functions, is then used to calculate the inclusive prefix sum of \mathbf{B} , denoted \mathbf{B}^{pre} . Each element $\mathbf{B}_{i,j}^{\text{pre}}$ contains the sum of elements of \mathbf{B} up to and including $\mathbf{B}_{i,j}$, given the fact that \mathbf{B} is stored as the concatenation of its rows in GPU memory. This means that the sum of a row is equal to the last element of this row minus the last element of the previous row. As a result of this, the value of s_j^{g+1} of P_j can be calculated in constant time through:

$$s_j^{g+1} = s_j^g \oplus \left(\mathbf{B}_{k,2j-1}^{\text{pre}} \ominus \mathbf{B}_{k,2j-2}^{\text{pre}} \right) \ominus \left(\mathbf{B}_{k,2j}^{\text{pre}} \ominus \mathbf{B}_{k,2j-1}^{\text{pre}} \right).$$

4.3 Hybridization

It is important to realize that the best performing parallelization technique can be different for partial evaluations of different complexities. Because of this, we write the time required for a partial evaluation of k sub-functions as $T = \alpha + k\beta$, where α indicates the

$f_1^M(\mathbf{p}_1^{g+1} _{I_1})$	$f_2^M(\mathbf{p}_1^{g+1} _{I_2})$...	$f_k^M(\mathbf{p}_1^{g+1} _{I_k})$
$f_1^M(\mathbf{p}_1^g _{I_1})$	$f_2^M(\mathbf{p}_1^g _{I_2})$...	$f_k^M(\mathbf{p}_1^g _{I_k})$
$f_1^M(\mathbf{p}_2^{g+1} _{I_1})$	$f_2^M(\mathbf{p}_2^{g+1} _{I_2})$...	$f_k^M(\mathbf{p}_2^{g+1} _{I_k})$
$f_1^M(\mathbf{p}_2^g _{I_1})$	$f_2^M(\mathbf{p}_2^g _{I_2})$...	$f_k^M(\mathbf{p}_2^g _{I_k})$
⋮	⋮	⋮	⋮
$f_1^M(\mathbf{p}_n^{g+1} _{I_1})$	$f_2^M(\mathbf{p}_n^{g+1} _{I_2})$...	$f_k^M(\mathbf{p}_n^{g+1} _{I_k})$
$f_1^M(\mathbf{p}_n^g _{I_1})$	$f_2^M(\mathbf{p}_n^g _{I_2})$...	$f_k^M(\mathbf{p}_n^g _{I_k})$

Figure 1: Matrix \mathbf{B} , containing the results of the *map* phase for the new population P^{g+1} and its previous state P^g .

complexity of the *process* phase, and β indicates the complexity of the *map* and *reduce* phases, assuming that the complexity of each sub-function is roughly identical.

Based on k , α and β , the most appropriate parallelization technique should be selected at different points throughout optimization. For example, because a linkage tree is used in the optimization of HDR BT with MO-RV-GOMEA [17], variation of any parent solution can consist of the modification of any number between 1 and ℓ variables. In this optimization problem, the number of modified variables determines the complexity of each sub-function β .

Criteria that determine the optimal parallelization technique can be determined prior to optimization based on the required computation time of a partial evaluation for various values of k . Based on these criteria, a hybrid parallelization approach can be used during optimization, meaning that a different parallelization technique is selected for partial evaluations of different complexities.

5 EXPERIMENTS

We compare different parallelization techniques for applying partial evaluations to a population of solutions, given different settings for the number of parallel sub-functions of the evaluation function, and the required computation time of each sub-function. We compare the use of 8 CPU cores to an Intel Xeon Phi, and an NVIDIA Titan X GPU. The native mode was used for all experiments with the Intel Xeon Phi, because preliminary experiments showed that the Intel Xeon Phi offload mode was never superior to the GPU. This is caused by the fact that both techniques require memory transfer to the device, while the GPU has more computational power.

5.1 Experimental Set-up

Each experiment consists of the partial evaluation of a population of n solutions consisting of ℓ real-valued variables represented by single-precision (32 bit) floating point numbers. Each partial evaluation requires the calculation of k independent sub-functions, and each sub-function has a complexity β . Sub-function complexity β is presented as a number of operations c , meaning that the arbitrary function $i^{3,14}$ for $i \in \{1, 2, 3, \dots, c\}$ was performed to artificially increase the sub-function complexity.

All experiments using an Intel Xeon Phi co-processor are performed on a computer with a 5100 series Intel Xeon Phi using the Knight's Corner architecture, having 60 cores at 1.05 GHz. All remaining experiments are performed on a computer with 10 Intel Xeon CPU E5-2630 at 2.2 GHz, and an NVIDIA Titan X GPU. Parallelization on multiple CPU cores is implemented in OpenMP. Each reported result is the average of 1000 independent runs.

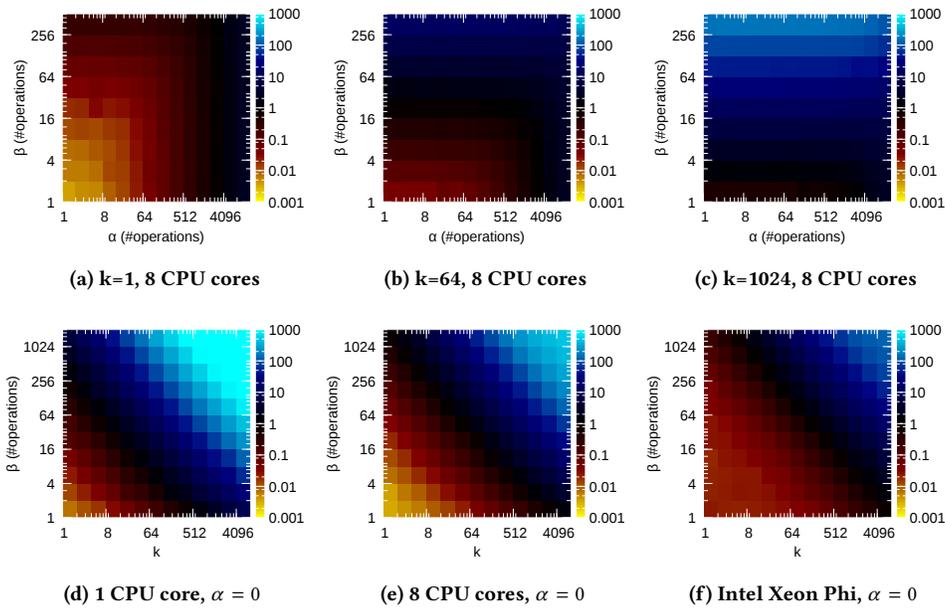


Figure 2: Speed-ups achieved by the GPU compared to other parallelization techniques, for different values of the number of sub-functions k and the sub-function complexity indicators α and β .

5.2 Experimental Results

In Figures 2a to 2c we show speed-ups achieved by the GPU compared to 8 CPU cores, given different settings for k , α , and β . The color of each block indicates the speed-up given the parameters of the block’s bottom-left corner. A population size of $n = 128$ is used, as preliminary experiments showed that achievable speed-ups were similar among different settings for the population size.

Figures 2a to 2c show that the effect of increasing α is only apparent for small values of k . Furthermore, for $k = 1$ and $k = 64$, an increase in α still leads to improved relative performance of the GPU, due to its larger number of cores. Despite this, an effective slow-down is observed due to overhead caused by memory transfer, and the GPU being unable to fully utilize its processing power. Finally, Figure 2c shows that a sufficiently large β is required to achieve any speed-up, caused by the overhead of copying memory to the GPU and back.

Figures 2d to 2f show the speed-ups achieved by the GPU compared to 1 CPU core, 8 CPU cores, and the Intel Xeon Phi, for a population size of $n = 128$, and $\alpha = 0$. We clearly see that the GPU performs very well at computing a large number of sub-functions with a high complexity. For 4096 sub-functions with $\beta = 1024$, the GPU achieved a speed-up of a factor 2803 compared to a single CPU core, a speed-up of a factor 446 compared to 8 CPU cores, and a speed-up of a factor 121 compared to the Intel Xeon Phi.

Figure 3 shows the best performing parallelization technique for different settings of the number of sub-functions and their complexity β , with CPU_m denoting parallel execution on m CPU cores, and XPhi denoting the Intel Xeon Phi. The left figure displays the sub-function complexity in terms of the number of operations, and the right figure displays the sub-function complexity in terms of the average computation time of a sub-function, which is calculated by dividing the average required computation time of 1 CPU core

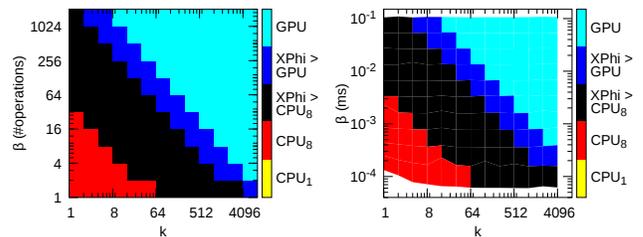


Figure 3: Best parallelization technique for different values of the number of sub-functions k and their complexity β .

by kn . In scenarios where the Intel Xeon Phi resulted in the best performance, the second best performing parallelization technique is also displayed, as this gives more insight that can be used for a hybrid approach using only a GPU and a number of CPU cores.

5.3 Discussion

The optimal technique for the parallelization of a partial evaluation clearly depends on factors including the number of sub-functions and their computational complexities. Figure 3 shows that the GPU performs very well at problems with a large number of sub-functions. In the largest-scale setting of this figure, with $k = 4096$ and $\beta = 0.1$ ms, the serial evaluation of one solution still only takes 0.4 seconds for small values of α . Performing a reasonable number of 10^5 evaluations of this complexity then takes roughly 11 hours, which could be reduced to a time of less than a minute by using large-scale parallelization on a GPU, assuming little global memory access is required. We therefore see a lot of potential in the use of a GPU for the large-scale parallelization of real-world problems in GBO settings.

Table 1: Clinical protocol for HDR BT at the AMC, with dose thresholds defined as a percentage of the planning-aim dose of 13 Gy.

Prostate	Bladder	Rectum	Urethra	Vesicles
$V_{100} > 95\%$	$D_{1cm^3} < 86\%$	$D_{1cm^3} < 78\%$	$D_{0.1cm^3} < 110\%$	$V_{80} > 95\%$
$V_{150} < 50\%$	$D_{2cm^3} < 74\%$	$D_{2cm^3} < 74\%$		
$V_{200} < 20\%$				

6 BRACHYTHERAPY

We now consider the optimization of BT treatment plans as described in [17]. Recently, experimental results showed that MO-RV-GOMEA [5] using partial evaluations achieved the best performance among a set of state-of-the-art MOEAs when using a single CPU. Furthermore, the formulation of this real-world problem seems well-suited for large-scale parallelization.

BT is a type of cancer treatment where cancerous tissue is irradiated from inside the body of the patient. We specifically address the HDR BT treatment for prostate cancer. This type of treatment consists of the insertion of 14-20 catheters into the body of a patient through the transperineal skin. These catheters are placed under general anaesthetics into the prostate and the seminal vesicles, which are the target volumes of this treatment. The Organs At Risk (OARs) surrounding the prostate, i.e., the bladder, rectum, and urethra, should receive as little radiation as possible. Radiation is delivered to the target volumes by passing a radioactive source through the catheters, and stopping this radioactive source at certain *dwell positions* for a certain amount of time, i.e., the *dwell time*. The longer a certain dwell time, the higher the delivered radiation dose to the tissue surrounding the dwell position.

In order to create a treatment plan, Magnetic Resonance Imaging (MRI) scans are taken of the patient after catheter placement. A clinician then contours the target volumes and the OARs, and specifies the locations of the catheters on the MRI scans. As a treatment plan is uniquely defined by its dwell times, these dwell times are often optimized using inverse planning algorithms [12]. At our collaborative medical center, the Academic Medical Center (AMC) in Amsterdam, this is followed by manual optimization by a clinician.

The quality of a treatment plan is assessed by visual inspection of the dose distribution, and in terms of a clinical protocol. A clinical protocol includes a set of planning criteria that describe clinically acceptable upper/lower bounds for the radiation dose delivered to the target volumes and the OARs. The clinical protocol at the AMC consists of a set of planning criteria, as listed in Table 1, defined in terms of a dose-volume-index (DVI) and a threshold.

The volume-index V_d^o is the volume of the organ o that receives at least $d\%$ of the planning-aim dose. For example, the planning criterion $V_{100}^{\text{prostate}} > 95\%$ indicates that at least 95% of the prostate volume should receive at least 100% of the planning-aim dose. A dose-index D_v^o specifies the minimum dose delivered to the most irradiated subvolume v of organ o . For example, the planning criterion $D_{1cm^3}^{\text{bladder}} < 86\%$ indicates that the minimum dose in the most irradiated 1 cm^3 of the bladder should be less than 86% of the planning-aim dose.

DVIs of a treatment plan t are estimated with Monte Carlo sampling, using a vector d describing the dose received by the set of

dose calculation points. Increasing the number of dose calculation points leads to more accurate estimation of DVIs, but increases computation time. The total dose received at a certain dose calculation point is the sum of the dose received from each dwell position. A pre-processing step is performed to compute a *dose-rate matrix* R [18], where $R_{i,j}$ defines the dose contribution (in Gy/s) of the dwell position corresponding to t_i , to the dose calculation point d_j . As such, d is calculated as $d = Rt$.

A dose-index D_v^o is estimated by first sorting the points in d that are inside organ o in descending order, denoted as $d^{s,o}$. The dose-index D_v^o is then equal to the dose point with index $\lfloor v/v^o \rfloor$ in $d^{s,o}$, i.e., $D_v^o = d_{\lfloor v/v^o \rfloor}^{s,o}$, where v^o is the volume corresponding to each point in organ o . A volume-index V_d^o is estimated by calculating the fraction of points in o that receive a dose of at least d . Given d , the calculation of a dose-index therefore has time complexity $\mathcal{O}(|d| \log |d|)$, and the calculation of a volume-index has time complexity $\mathcal{O}(|d|)$.

6.1 Treatment Plan Optimization

Because the OARs are in close proximity of the target volumes, with the urethra even passing right through the prostate, there is a clear trade-off between covering the target volumes and sparing the OARs. The aim of HDR BT treatment planning is then to find dwell times such that an acceptable trade-off between target coverage and sparing is achieved. Multi-objective EAs are among the state-of-the-art for multi-objective optimization [10], and they result in a large set of solutions having high-quality trade-offs between the objectives of interest. From this set of solutions, the most appropriate treatment plan can then be selected *a posteriori* by a clinician. Among other reasons, this is why a multi-objective optimization approach was used in [17]. However, such an approach is relatively time-consuming, while limited time is available in clinical practice.

Because a set of dwell times uniquely defines a treatment plan, each solution in the population is described by a vector of dwell times t of length ℓ . The planning criteria $V_{150}^{\text{prostate}} < 50\%$, and $V_{200}^{\text{prostate}} < 20\%$ are used as hard constraints for the optimization, i.e., treatment plans not satisfying these criteria are considered infeasible. The remaining set of planning criteria is condensed into two objectives: the Least Coverage Index (LCI) and the Least Sparing Index (LSI), indicating how close the DVIs of a treatment plan are to the thresholds defined by the clinical protocol. LCI and LSI are defined such that positive values indicate that all planning criteria are satisfied with regard to coverage and sparing, respectively.

The LCI of a treatment plan t is defined as

$$\text{LCI}(t) = \min \left\{ \delta_v(V_{100}^{\text{prostate}}), \delta_v(V_{80}^{\text{vesicles}}) \right\},$$

$$\delta_v(V_d^o) = V_d^o - V_d^{o,\min},$$

with $V_d^{o,\min}$ the threshold defined by the clinical protocol.

The LSI of a treatment plan t is defined as

$$\text{LSI}(t) = \min \left\{ \delta_d(D_{1cm^3}^{\text{bladder}}), \delta_d(D_{2cm^3}^{\text{bladder}}), \right. \\ \left. \delta_d(D_{1cm^3}^{\text{rectum}}), \delta_d(D_{2cm^3}^{\text{rectum}}), \delta_d(D_{0.1cm^3}^{\text{urethra}}) \right\},$$

$$\delta_d(D_v^o) = D_v^{o,\max} - D_v^o,$$

with $D_v^{o,\max}$ the threshold defined by the clinical protocol.

Partial evaluations can be used to efficiently update \mathbf{d} if only a subset of elements of \mathbf{t} is modified. Given a dose distribution and a change in dwell times $\Delta\mathbf{t}$, the effect of this change in dwell times, resulting in \mathbf{d}' , can be efficiently computed as

$$\begin{aligned}\mathbf{d}' &= \mathbf{d} + \Delta\mathbf{d} \\ &= \mathbf{d} + \mathbf{R}\Delta\mathbf{t}.\end{aligned}$$

For each element $\Delta\mathbf{t}_i = 0$, the column $\mathbf{R}_{i,*}$ can be skipped in the calculation of $\Delta\mathbf{d}$. This means that $\Delta\mathbf{d}$ can be calculated in $O(k|d|)$ time given a vector $\Delta\mathbf{t}$ with k non-zero elements, as opposed to $O(\ell|d|)$ for the calculation of \mathbf{d} . For each solution in the population, the dose distribution \mathbf{d} is maintained in memory, allowing it to be updated efficiently. This requires $O(n|d|)$ memory for a population of size n .

6.2 GBO Parallelization

We now apply large-scale GBO parallelization, as introduced in Section 4, to HDR BT treatment plan optimization. In this application, the calculation of the two objectives, LCI and LSI, relies on the intermediate step of the calculation of \mathbf{d} . We focus on the parallelization of the calculation of \mathbf{d} , because this is the part of the evaluation where partial evaluations can be applied, and where sub-functions can be identified.

As discussed in Subsection 6.1, partial evaluations can be applied by calculating $\Delta\mathbf{d}$ and adding it to \mathbf{d} . The calculation of each element \mathbf{d}_j can be considered a function to which partial evaluations can be applied, described as a sum of sub-functions, as follows:

$$\mathbf{d}_j = \sum_{i=0}^{\ell-1} \mathbf{R}_{i,j} \mathbf{t}_i.$$

As such, the number of partial evaluations that can be computed in parallel is equal to $|d|$, where the complexity of one partial evaluation of k variables is equal to $O(k)$. Given the fact that all solutions in the population can be evaluated in parallel, this leads to a total of $n|d|$ operations that can be performed in parallel.

In the following way, the phases described in Section 4 are applied to HDR BT treatment plan optimization:

- (1) *Map*
 - In parallel for each $\Delta\mathbf{t}_i \neq 0$, calculate $\mathbf{R}_{i,j}\Delta\mathbf{t}_i$.
- (2) *Reduce*
 - In parallel for each \mathbf{d}_j , calculate $\sum_{\Delta\mathbf{t}_i \neq 0} \mathbf{R}_{i,j}\Delta\mathbf{t}_i$.
- (3) *Process*
 - $\mathbf{d}' = \mathbf{d} + \Delta\mathbf{d}$
 - Calculation of DVIs; time complexity $O(|d| \log |d|)$.

The above phases are only synchronized between all solutions in the population when GPU parallelization is used. Note that the combined *map* and *reduce* phases come down to the matrix-vector multiplication $\mathbf{R}\Delta\mathbf{t}$. Due to the synchronization on the GPU, this can be done in parallel for the entire population given the matrix \mathbf{T} , with $\mathbf{T}_{*,k}$ the vector $\Delta\mathbf{t}$ for solution \mathbf{P}_k , resulting in the single matrix-matrix multiplication $\mathbf{R}\mathbf{T}$ being sufficient for the *map* and *reduce* phases of the entire population.

6.3 Experiments

We now compare different parallelization techniques applied to partial evaluations with different complexities, depending on the

number of modified dwell times and the number of dose calculation points, for HDR BT treatment plan optimization. Typically, 1000 dose calculation points are used for inverse planning approaches [15], and 20000 points are used for multi-objective optimization [17]. Because the *process* phase is relatively time consuming, GPU parallelization is always applied to this phase when possible. This is however not possible when the Intel Xeon Phi is used in native mode, because all computation is done on the Intel Xeon Phi, and the host machine that can also host a GPU, is not directly used.

In Figures 4a to 4c we show the speed-ups achieved by the GPU compared to the three other parallelization techniques for a population size of 112. This population size was selected, because a population size of approximately 100 was found to be sufficient for the optimization, a population size divisible by 8 is preferred for the parallelization on 8 cores, and a population size divisible by 16 is preferred for GPU parallelization, due to the dimensions of the grid discussed in Section 3.

We observe that the GPU performs slightly worse than both 1 CPU core and 8 CPU cores for the setting of 1000 dose calculation points and 1 modified dwell time, by factors 0.74 and 0.68 respectively. For 128000 dose calculation points and 220 modified dwell times, the GPU achieves a speed-up of a factor 73 compared to a single CPU core, and a speed-up of a factor 12 compared to 8 CPU cores. A decrease in speed-up compared to the Intel Xeon Phi appears for a large number of dose calculation points and modified number of dwell times. This is caused by the fact that the *process* phase is not performed on a GPU when the Intel Xeon Phi is used, leading to a relatively slow performance of the Intel Xeon Phi when a large number of dose calculation points is used, but a small number of dwell times is modified.

The best performing parallelization techniques for different settings are displayed in Figure 4d, with CPU_n denoting parallel execution on n CPU cores, and XPhi denoting the Intel Xeon Phi. We observe that the Intel Xeon Phi performs best for a small number of dose calculation points and modified dwell times, but is quickly overtaken by the GPU when over 4000 points are used.

6.4 Discussion

We observed that the GPU is only outperformed by different parallelization techniques when up to 2000 dose calculation points are used. However, it was previously shown that high-quality treatment plans were unable to be found when only 4000 dose calculation points were used [17]. This shows that GPU parallelization is very well suited for HDR BT treatment plan optimization.

7 CONCLUSIONS

We have introduced an approach for the large-scale parallelization of partial evaluations in GBO, for example using a GPU or a many-core architecture such as the Intel Xeon Phi, which evaluates a large number of sub-functions of the evaluation function in parallel. This approach is aimed at real-world problems with time-consuming evaluations to which GBO can be applied. We analyzed the performance of the various parallel co-processor architectures in a large number of scenarios depending on, for example, the number of sub-functions of a partial evaluation that can be computed in parallel, and the complexity of such sub-functions. We note that, if partial

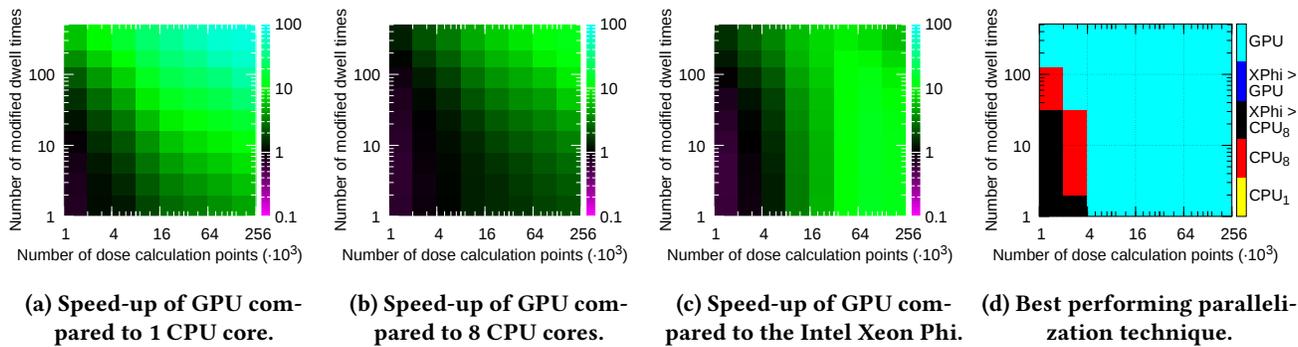


Figure 4: Results on the parallelization of HDR BT partial evaluations, showing speed-ups achieved by the GPU and the best parallelization technique given different parameter settings. A different color scheme is intentionally used, because speed-ups are shown on a different scale than before.

evaluations are possible, EAs can likely benefit from hybrid parallelization approaches, where the most appropriate parallelization approach is selected for various parts of the optimization process. Furthermore, we conclude that large-scale parallelization is likely to be applicable to time-consuming real-world problems that can be solved in a GBO setting. Applying GPU parallelization resulted in speed-ups of factors over 2000 in the most favorable scenarios of our experiments.

We have applied the large-scale parallelization model to HDR BT treatment plan optimization, and we have found that parallelization on a GPU outperforms other parallelization techniques in all scenarios with realistic parameter settings, resulting in speed-ups of the partial evaluations by up to a factor of 73. These are very promising results that bring us closer to the application of multi-objective optimization of HDR BT treatment plans in a time-constrained clinical setting.

8 ACKNOWLEDGMENTS

This work is part of the research programme IPPSI-TA with project number 628.006.003, which is financed by the Netherlands Organisation for Scientific Research (NWO) and Elekta. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

REFERENCES

- [1] N. Bell and J. Hoberock. 2011. Thrust: A productivity-oriented library for CUDA. *GPU computing gems Jade edition 2* (2011), 359–371.
- [2] P. A. N. Bosman and D. Thierens. 2011. The roles of local search, model building and optimal mixing in evolutionary algorithms from a BBO perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 663–670.
- [3] A. Bouter, T. Alderliesten, and P. A. N. Bosman. 2017. A novel model-based evolutionary algorithm for multi-objective deformable image registration with content mismatch and large deformations: benchmarking efficiency and quality. *Proc.SPIE* 10133, 1013312.
- [4] A. Bouter, T. Alderliesten, C. Witteveen, and P. A. N. Bosman. 2017. Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 705–712.
- [5] A. Bouter, N. H. Luong, C. Witteveen, T. Alderliesten, and P. A. N. Bosman. 2017. The multi-objective real-valued gene-pool optimal mixing evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 537–544.
- [6] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra. 2013. Graphics processing unit (GPU) programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing* 73, 1 (2013), 4–13.
- [7] F. Chicano, D. Whitley, G. Ochoa, and R. Tinós. 2017. Optimizing one million variable NK landscapes by hybridizing deterministic recombination and local search. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 753–760.
- [8] NVIDIA Corporation. 2018. CUDA C Programming guide v9.1.85. (2018).
- [9] D. D’Agostino, G. Pasquale, and I. Merelli. 2014. A Fine-Grained CUDA Implementation of the Multi-objective Evolutionary Approach NSGA-II: Potential Impact for Computational and Systems Biology Applications. In *International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*. Springer, 273–284.
- [10] K. Deb. 1999. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation* 7, 3 (1999), 205–230.
- [11] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.
- [12] A. M. Dinkla, R. van der Laarse, E. Kaljouw, B. R. Pieters, K. Koedooder, N. van Wieringen, and A. Bel. 2015. A comparison of inverse optimization algorithms for HDR/PDR prostate brachytherapy treatment planning. *Brachytherapy* 14, 2 (2015), 279–288.
- [13] N. Hansen, S. D. Müller, and P. Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation* 11, 1 (2003), 1–18.
- [14] J. Jeffers, J. Reinders, and A. Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.
- [15] E. Lessard and J. Pouliot. 2001. Inverse planning anatomy-based dose optimization for HDR-brachytherapy of the prostate using fast simulated annealing algorithm and dedicated objective function. *Medical physics* 28, 5 (2001), 773–779.
- [16] S. C. Li and T. L. Yu. 2017. Speeding up DSMGA-II on CUDA platform. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 809–816.
- [17] N. H. Luong, T. Alderliesten, A. Bel, Y. Niatsetski, and P. A. N. Bosman. 2017. Application and benchmarking of Multi-Objective Evolutionary Algorithms on High-Dose-Rate brachytherapy planning for prostate cancer treatment. *Swarm and Evolutionary Computation* (2017). <https://doi.org/10.1016/j.swevo.2017.12.003>
- [18] R. Nath, L. L. Anderson, G. Luxton, K. A. Weaver, J. F. Williamson, and A. S. Meigooni. 1995. Dosimetry of interstitial brachytherapy sources: recommendations of the AAPM Radiation Therapy Committee Task Group No. 43. *Medical physics* 22, 2 (1995), 209–234.
- [19] G. Ortega, E. Filatovas, E. M. Garzón, and L. G. Casado. 2017. Non-dominated sorting procedure for pareto dominance ranking on multicore CPU and/or GPU. *Journal of Global Optimization* 69, 3 (2017), 607–627.
- [20] D. Thierens and P. A. N. Bosman. 2011. Optimal mixing evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 617–624.
- [21] R. Tintos, D. Whitley, and F. Chicano. 2015. Partition crossover for pseudo-boolean optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*. ACM, 137–149.
- [22] D. Whitley. 2017. Next generation genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 922–941.
- [23] M. L. Wong and T. T. Wong. 2009. Implementation of parallel genetic algorithms on graphics processing units. *Intelligent and Evolutionary Systems* 187 (2009), 197–216.