Silhouetting the Cost-Time Front: Multi-objective Resource Optimization in Business Processes

Orlenys López-Pintado[1], Marlon Dumas[1(✉)], Maksym Yerokhin[1], and Fabrizio Maria Maggi[2]

[1] University of Tartu, Tartu, Estonia
{orlenys.lopez.pintado,marlon.dumas,maksym.yerokhin}@ut.ee
[2] Free University of Bozen-Bolzano, Bolzano, Italy
maggi@inf.unibz.it

Abstract. The allocation of resources in a business process determines the trade-off between cycle time and resource cost. A higher resource utilization leads to lower cost and higher cycle time, while a lower resource utilization leads to higher cost and lower waiting time. In this setting, this paper presents a multi-objective optimization approach to compute a set of Pareto-optimal resource allocations for a given process concerning cost and cycle time. The approach heuristically searches through the space of possible resource allocations using a simulation model to evaluate each allocation. Given the high number of possible allocations, it is imperative to prune the search space. Accordingly, the approach incorporates a method that selectively perturbs a resource utilization to derive new candidates that are likely to Pareto-dominate the already explored ones. The perturbation method relies on two indicators: resource utilization and resource impact, the latter being the contribution of a resource to the cost or cycle time of the process. Additionally, the approach incorporates a ranking method to accelerate convergence by guiding the search towards the resource allocations closer to the current Pareto front. The perturbation and ranking methods are embedded into two search metaheuristics, namely hill-climbing and tabu-search. Experiments show that the proposed approach explores fewer resource allocations to compute Pareto fronts comparable to those produced by a well-known genetic algorithm for multi-objective optimization, namely NSGA-II.

Keywords: Business process optimization · Resource allocation · Multi-objective optimization · Process simulation

1 Introduction

A business process brings together several activities performed by participants (a.k.a. resources) that are typically divided into groups (a.k.a. resource pools) according to their areas of responsibility. Each resource pool has a capacity,

© The Author(s) 2021
A. Polyvyanyy et al. (Eds.): BPM Forum 2021, LNBIP 427, pp. 92–108, 2021.
https://doi.org/10.1007/978-3-030-85440-9_6

determined by the number of resources in the pool. For example, in a loan application handling process, there may be a resource pool grouping multiple clerks responsible for all activities related to collecting and validating data, a Credit Officer pool responsible for preparing initial loan decisions, and a Senior Credit Officer pool for validating these decisions and handling exceptional cases.

The problem of resource allocation is that of determining how much capacity (i.e., how many resources) to allocate to each resource pool so as to minimize or maximize one or more performance measures. In this respect, for a given workload, the more there are resources in a pool, the less busy these resources are (low resource utilization). Conversely, the less there are resources in a pool, the busier the resources are (high resource utilization). Higher resource utilization leads to lower cost per instance (as resources are used to their full extent) and high waiting times (due to resource contention). Conversely, lower resource utilization leads to higher cost per instance and to lower waiting times [10]. Managers need to balance these two ends of the spectrum, aiming for a resource allocation that minimizes both the costs and the waiting times. Typically, no single solution exists that minimizes time and cost simultaneously. Instead, there is a set of (incomparable) optimal solutions (a.k.a. Pareto front) so that no objective, e.g., time and cost, can be improved without scarifying any other.

This paper presents an approach to compute a set of Pareto-optimal resource allocations for a business process. The approach iteratively explores the space of possible resource allocations and uses the simulation model to assess the cost and cycle time of each explored allocation.[1] The search space is traversed using hill-climbing and tabu-search meta-heuristics. In each iteration, we estimate the resource utilization and the overall resource performance (i.e., their impact on the cost-time space), and we use these indicators to guide a perturbation method that selects a subset of neighbors that are likely to Pareto-dominate solutions in the current Pareto front, instead of exploring the entire neighborhood of each allocation. The search strategy employs a ranking method to prioritize new candidate allocations. Additionally, to cater for the fact that the output of a simulation model is subject to stochastic variations, we propose a notion of Pareto-dominance based on the median absolute deviation of the simulation outputs. These mechanisms lead to two enhanced variants of hill-climbing plus an enhanced variant of tabu-search. The paper reports an experimental evaluation to assess the convergence, spread, and distribution of the discovered Pareto fronts and the number of explored resource allocations, relative to a well-known genetic algorithm for multi-objective optimization (NSGA-II) [8].

The rest of the paper is structured as follows. Section 2 discusses related works. Section 3 introduces key concepts and meta-heuristics for multi-objective optimization and concepts related to process simulation. Section 4 describes the perturbation and ranking methods and the enhanced hill-climbing and

[1] In the experiments, we use simulation models discovered from execution data, but the approach can take as input a manually designed simulation model or any stochastic model capable of estimating costs and cycle times for different resource allocations.

tabu-search variants. Then, Sect. 5 discusses the implementation and evaluation, while Sect. 6 concludes the paper.

2 Related Work

Several previous studies have addressed the problem of resource allocation in business processes. However, the bulk of these studies addressed resource allocation as a single-objective optimization problem, i.e., either by optimizing one performance measure or combining several into a linear function [13,16,19,23].

In [19], the authors proposed an evolutionary algorithm for finding the optimal resource allocation of a business process. The framework's input is a Colored Petri Net, including all the parameters necessary for simulation, such as arrival rate, processing times for each task, and branching probabilities for each decision point. The paper optimizes the resource allocation regarding cycle time and cost, combined into a single performance measure through a linear function. Similar approaches using genetic algorithms and simulation models on single objective-problems were presented in [9,14]. In the present paper, instead of combining the time and the cost, we compute an entire Pareto front, which allows the user to explore the available trade-offs between cycle time and resource cost.

The work presented in [16] addresses the optimization problem as an exploration of the space of possible resource allocations. The approach considers the resource utilization to define three strategies to discover the optimal resource allocation while performing a reduced search of the solution candidates. The authors addressed the resource allocation as a single-objective optimization problem, i.e., minimizing the number of resources constrained by a specified maximum waiting time. This paper adopts a different approach that considers resource utilization in a multi-objective optimization setting to discover not a single optimal but a set of optimal solutions.

In [11], the authors analyze the relationship between resource allocation and various performance measures, including time. The authors use a grid-search approach, i.e., an exhaustive exploration of all possible resource allocations given a minimum and a maximum number of resources per pool. This approach can be applied to explore the resource allocation space when the number of pools is small. However, it does not scale up to larger search spaces.

The problem of design-time resource allocation tackled in the present paper is related to the problem of runtime scheduling and runtime assignment of resources to work items in a business process. The latter problems have been tackled in various previous studies. For example, [18] and [22] consider the problem of deciding how to schedule the work items generated by each execution of a business process, taking into account that resources have availability constraints (i.e., they are available at some times but not at others). Meanwhile, [12] tackles the problem of deciding which specific resource should be assigned to a given work item, given the characteristics of each resource. The contribution of the present paper and those of the above papers are complementary. After selecting a given resource allocation using the techniques proposed in this paper, it is

perfectly possible to optimize the runtime scheduling and assignment of resources to work items using the techniques developed in the above papers.

The problem of resource allocation has also been studied outside the field of business process optimization. For example, in [6], the authors present an algorithm to discover Pareto fronts relying on ant colony optimization, assessing several performance measures for a given resource allocation. From this latter study, we share the idea of formulating the resource allocation as a multi-objective problem but adapted to the meta-heuristics hill-climbing and tabu-search.

3 Overview of Multi-objective Optimization and Business Process Simulation

3.1 Pareto Fronts and Meta-heuristic Optimization Algorithms

In an n-dimensional space, a solution B is Pareto dominated by another solution A, if A is better than B for at least one objective, and A is at least as good as B for the remaining objectives [2], e.g., $B = (2, 5, 10)$ and $C = (3, 8, 12)$ are Pareto dominated by $A = (1, 5, 10)$, under minimization constraints. The set of solutions that are not dominated by any other are called Pareto optimal. The set of non-dominated points are called the Pareto set, and the evaluation of the objective functions on those points constitutes the Pareto front [2]. For example, in the two-dimensional space cost-time associated with the execution of a business process, the Pareto set contains the resource pools whose respective cost-space evaluations constitute a Pareto front, i.e., the pairs cost-time are Pareto-optimal. However, the problem of resource allocation is a well-known NP-complete problem. Thus, as no efficient solution exists (i.e., exploring the entire solution space is not possible in practical scenarios), some meta-heuristic algorithms can be used to approximate the Pareto fronts.

Among many other classifications, existing meta-heuristic optimization algorithms can be broadly classified into single-solution-based and population-based. Single-solution algorithms keep one solution and search for better solutions at each step through a perturbation function. Population-based algorithms keep a population of solutions and build a new population at each step by perturbing and combining solutions in the existing population. Indeed, single-solution approaches are more efficient (i.e., they explore a lower number of solutions), but population-based techniques lead to more optimal solutions at the cost of exploring a higher number of solution candidates [4]. This paper focuses on enhancing two of the most well-known single-solution-based meta-heuristics, named hill-climbing and tabu-search. Besides, we use one population-based approach, the genetic algorithm NSGA-II, as a baseline in our experiments.

Hill-climbing is an optimization technique that performs a local search around a given point. At each iteration, the algorithm selects the best possible point to move in the current point neighborhood. Therefore, the algorithm improves the current solution on each iteration unless the entire neighborhood does not contain better solutions. Classic applications of this algorithm assume a single

objective [4] (e.g., time, cost, or a linear combination of both). However, in [20], the authors describe a modification of hill-climbing for multi-objective optimization (i.e., to compute a Pareto front). To that end, not a single solution but a Pareto front is stored. Thus, the new solution candidates are generated by taking each point in the current front and generating its neighborhood. The greedy nature of hill-climbing allows it to converge fast, but it may stop at a local optimum.

The tabu-search algorithm is an extension of hill-climbing that avoids the limitation of getting stuck in a local optimum. Unlike hill-climbing, tabu-search stores the current best point, but it also accepts inferior solutions if no improvement is found from the current best solution's neighborhood. Thus, it accepts Pareto-dominated solutions temporarily to visit new parts of the search space, aiming to converge to the global optimum in subsequent iterations. The implementation includes a so-called tabu list, so solutions already visited or restricted by any other rules are marked as *tabu*, thus not revisited (i.e., at least in a short-term period) [4]. Although classical variants of the tabu-search algorithm assume a single-objective, like hill-climbing, it can be extended to a multi-objective space by considering a Pareto front instead of a single solution [20].

NSGA-II, the acronym of Non-dominated Sorting Genetic Algorithm, is a well-known genetic algorithm designed explicitly for multi-objective optimization [8]. The algorithm's idea is to keep a population of points, some of which are in the Pareto front, and others are not but well placed along with one of the dimensions. At each iteration, the algorithm generates off-springs by sampling from the neighborhood of the points in the current population. The best new solutions are added to the population, and a subset of the existing solutions (which are not Pareto optimal) are removed. To determine which solutions to add or remove, the algorithm measures how far the solutions in the current population are separated from each other.

3.2 Resource Pools, Event Logs and Business Process Simulation

A resource allocation is a sequence of resource pools $R = \{r_1, ..., r_n\}$, each responsible for a subset of activities in a process. The functions $rCount : R \rightarrow \mathbb{N}+$ and $rCost : R \rightarrow \mathbb{R}+$ retrieve, respectively, the number of resources and cost (per time unit) of using one resource in a pool r_i.

An event is a tuple $e =< \lambda, r, \gamma_s, \gamma_c >$, where λ is the label of one activity in a business process (i.e., e is an instance of the activity λ), $r \in R$ is the resource who performed λ, and γ_s, γ_c are, respectively, the time-stamps corresponding to the beginning and end of the event. A trace (a.k.a. process case) is a non-empty sequence of events $t =< e_1, e_2, ..., e_n >$, and an event log $eLog =< t_1, t_2, ..., t_m >$ is a non-empty sequence of traces corresponding to the execution of a process.

A simulation model consists of a process model M, e.g., written in the Business Process Model and Notation (BPMN) notation, a set of resource pools R, and a function $activityResource : A \rightarrow R$ that maps each activity $a \in A$ in the process model to a resource $r \in R$. Simulation models also include the mean

inter-arrival time of cases and probability distributions for arrival cases, activities' processing times, and gateways' branching [5]. Simulation models can be executed using simulation engines like BIMP [1], which produces a set of possible execution traces used to perform quantitative analysis of business processes. Henceforth, we will use the notation *rpLog* referring to event logs obtained from real executions of business process and the notation *smLog* to point out simulated event logs. In our approach, we consider the following functions computed from an event log *eLog*:

- $eventDuration(e =< \lambda, r, \gamma_s, \gamma_c >, eLog)$ represents the time-span, $\gamma_c - \gamma_s$, between the beginning and end of event e, (a.k.a. processing time), plus the time-span from the moment activity λ is enabled until the starting of the corresponding event (a.k.a. waiting time),
- $traceDuration(t, eLog)$ and $procDuration(eLog)$ retrieve the time-span between the beginning and end of trace t and the entire process, respectively,
- $cTime(R, eLog)$, i.e., cycle time, computes the average $traceDuration$ of all the traces $t \in eLog$, involving the resource pools in R,
- $aCost(R, eLog) = procDuration(eLog) * \sum_{r \in R}[rCost(r) * rCount(r)]$ corresponds to the cumulative costs of all the resources during the process execution. These costs consider not only the resources which performed each event $e \in eLog$ but all the resources allocated to the resource pools, which must be available at any time of the execution,
- $resourceUtilization(r, eLog)$ divides the time in which resources in pool r were busy by $procDuration(eLog)$, i.e., the percentage of time in which the resources are busy.

4 Computing the Pareto-Optimal Resource Allocations

4.1 Initial Resource Allocation and Process Simulation

To discover the simulation model from an event log *rpLog* provided as input, we use the tool named Simod [5]. It produces a process model in BPMN extended with the probability distributions of each element/branch. Besides, it provides the initial resource allocation R_0 and the mapping function *activityResource*. The incoming iterations produce only new resource allocations $R_1, R_2, ..., R_n$, i.e., the control-flow of the BPMN model, and the mapping *activityResource* remain unaltered. Henceforth, we will describe the steps of our approach based on the corresponding resource allocation (a.k.a. solution candidate) R_i.

For each resource allocation R_i, the evaluation of the objective functions $cTime$ and $aCost$ requires to simulate the process, i.e., to assess the impact of the current allocation on the execution. Due to the simulations' stochastic nature, running a single simulation per allocation may lead to inaccurate evaluations. Thus, we run a number $smCount \geq 10$ of simulations, keeping the results from the simulated log *smLog* with median values of the function $cTime$. Also, we calculate the absolute median deviation (MAD) for both objective functions, i.e.,

$MAD = median(\{|F_M - f_1|, ..., |F_M - f_n|\})$, where $F_M = median(\{f_1, ..., f_n\})$ with $f_i = f(R, smLog_i), \forall 1 \leq i \leq n = smCount$, and $f \in \{aCost, cTime\}$.

The MAD serves to introduce a more strict Pareto dominance relation, considering the simulation results' variability. In the classical Pareto dominance relation, a resource allocation R_i dominates R_j ($R_i < R_j$) if R_i has a lower cycle time and cost than R_j, i.e., $f(R_i) \leq f(R_j), f \in \{cTime, aCost\}$. In a more strict dominance relation, R_i strongly dominates R_j ($R_i << R_j$) if $f(R_i) \leq f(R_j)$ and $|f(R_i) - f(R_j)| > min(MAD(f(R_i)), MAD(f(R_j)))$, $f \in \{aCost, cTime\}$. Thus, the cycle time and cost of R_i should be lower than R_j by a difference of at least the minimum MAD between the two objective functions. In other words, although R_i may dominate R_j, they are still close to discard R_j as a Pareto optimal solution due to the simulations' variance.

4.2 Perturbation Method: Generating Solution Candidates

Like any hill-climbing and tabu-search approaches, our proposal constructs the Pareto front incrementally. At each iteration, instead of exploring the entire neighborhood of the Pareto front like in traditional approaches[2], we heuristically select which resources might have a higher impact on the process execution. Specifically, we introduce a perturbation that relies on two criteria to decide which resource pool to improve, i.e., resource utilization and resource impact.

We hypothesize that a high resource utilization may increase the cycle times, i.e., the resources are too busy, which might harm their overall performance. Thus, increasing the number of resources might lead to reducing the overall cycle time. Conversely, low resource utilization may affect the execution costs, i.e., there are some lazy resources with low efficiency, which might not be necessary. Thus, decreasing the number of resources may lead to a decrease in the execution costs without increasing the cycle times. Therefore, at each iteration, we select the pools with higher/lower resource utilization and accordingly add, remove or exchange resources to/from/between them.

Another issue to solve on the perturbation based on the resource utilization is the number of resources to add or remove. Adding/removing one resource leads to a shorter evolution step. Thus, it may increase the chances of finding a new allocation improving the current one, but it may require a high number of iterations to converge to the optimal. Conversely, adding/removing a higher number of resources to reach some desire utilization ratio may converge faster to the optimal allocation. Specifically, we use inverse proportion to estimate the amount of resources to add or remove by the formula:

$$amount = (resourceUtilization(r, smLog) * rCount(r)/dRu) - rCount(r) + 1$$

where dRu is a desired value for the resource utilization. In this paper, the perturbation function adds/removes the corresponding *amount* to/from the resource

[2] The neighborhood of a Pareto front might consist of 2^{mn} solutions (i.e., straightforwardly adding/removing one to/from each pool), where m and n are the size of the Pareto front and the number of resource pools, respectively. The latest makes the searching space too wide, especially when the number of resources is high.

Algorithm 1. Construction of the Pareto front

```
1: function AproximateParetoFront(rpLog)
2:     (SM, R_0) ← DiscoverSModel(rpLog)
3:     smLog_0 ← SimulateProcess(SM, R_0)
4:     PFront ← {< R_0, aCost(R_0, smLog_0), cTime(R_0, smLog_0) >}
5:     PriorityQ ← ∅
6:     Enqueue(PriorityQ, R_0, dist(R_0, PFront))
7:     while Q not empty do
8:         if StoppingCriteriaMet then
9:             return PFront
10:        SCandidates ← FindCandidates(Pop(PriorityQ))
11:        for each R_i ∈ SCandidates do
12:            smLog_i ← SimulateProcess(SM, R_i)
13:            if isNonDominated(R_i, smLog_i, PFront) then
14:                UpdateParetoFront(PFront, R_i, smLog_i)
15:                Enqueue(PriorityQ, R_i, dist(R_i, PFront))
16:            else if IsTabuSearch then
17:                Enqueue(PriorityQ, R_i, dist(R_i, PFront))
18:    return PFront
```

pools with higher/lower utilization to reach an ideal utilization, e.g., between 0.7–0.8 as Gartner analysts suggest, or any values set by the process analysts goals. The perturbation method also exchanges the minimum *amount* between the pools with higher and lower utilization. However, although the calculated *amount* introduces a higher step accelerating the convergence, it may also skip solution paths in the middle. Thus, the perturbation method uses both values unitary and the calculated *amount* to generate the next solution candidates.

To tackle other issues, i.e., not related to resource utilization, which may be harming the process performance, we use a heuristic considering the resources' impact. To that end, for each resource pool r, we calculate the aggregated costs and cycle times of the activities assigned to r, i.e., from the mapping function $activityResource$. Thus, the perturbation function will update the resource pools not improved from the previous heuristic regarding utilization, whose aggregated times and costs are above the average. Specifically, it increases the number of resources on pools showing higher cycle times since adding more resources may reduce the workload, thus decreasing the waiting times. Conversely, it reduces the number of resources on pools with higher costs since fewer resources performing the same activities more efficiently would reduce costs.

4.3 Ranking Method: Hill-Climbing and Tabu-Search Variants

Algorithm 1 sketches our proposal, which takes an event log as input. The steps in lines 2–3 discover the simulation model, the initial resource allocation R_0, and runs the first simulation, as described in Sect. 4.1. The initial Pareto front $Front$ contains the initial resource allocation discovered from the event log and the values $aCost$ and $cTime$ retrieved from the initial simulation.

A key difference of our approach with traditional variants of hill-climbing and tabu-search consists of sorting the solution candidates (i.e., resource allocations) based on their Euclidean distance to a Pareto front $PFront$:

$$dist(R_i, PFront) = \min_{p \in PFront} \|f(R_i) - f(p)\|_2 : f \in \{aCost, cTime\}.$$

Thus, the algorithm stores the solution candidates in a priority queue, which is initialized in lines 5–6.

At each iteration, the algorithm does not explore the neighborhood of each allocation in the current Pareto front. Instead, it uses the heuristics described in Sect. 4.2 to alter the solution candidate with the shortest Euclidean distance from the Pareto front (line 10). Next, it simulates the process for each allocation $R_i \in SCandidates$ retrieved by the perturbation method. Then, lines 11–17 verify, after evaluating the objective functions $aCost$ and $cTime$, if the allocation R_i is dominated (or not) by any allocation in the current Pareto front. Accordingly, a solution candidate is added to the Pareto front and the priority queue depending on the meta-heuristic search strategy as follows:

- **HC-STRICT**, or hill-climbing strict, considers the classical Pareto dominance relation. The resource allocation R_i is added to the Pareto front if it is Pareto-optimal, i.e., if the pair cost-time from R_i is not dominated by any of the pairs cost-time in $PFront$. Similarly, R_i is only added to the priority queue if it is Pareto-optimal. Also, each resource allocation dominated by R_i is discarded. Note that after updating $PFront$, the distance $dist$ to $PFront$ must be updated for each element in the priority queue.
- **HC-FLEX**, or hill-climbing flexible, considers a more strict Pareto dominance relation defined by the MAD, which produces a larger Pareto front. Thus, we relax the Pareto front definition to include classically dominated elements, i.e., those separated by at most the median absolute deviation for both objective functions $aCost$ and $cTime$. The steps to update the $PFront$ and the priority queue are the same as for $HC-STRICT$, but constructing a relaxed Pareto front.
- **TS-STRICT**, or tabu-search strict, uses the classical Pareto dominance relation. However, unlike hill-climbing, tabu-search also adds to the priority queue all the discarded, i.e., not Pareto-optimal resource allocations. So, when no Pareto-optimal allocation exists in the queue, the tabu-search will generate the subsequent solution candidates from the non-optimal resource allocation with the shortest distance to the current Pareto front.

The algorithm stops (lines 7–8) if any of the following conditions hold: (1) the queue is empty, (2) after exploring a specified maximum number of allocations, i.e., those generated by the perturbation function, (3) after producing a maximum number of consecutive Pareto non-optimal allocations. Then, $PFront$ is returned as approximation of the optimal resource allocation.

5 Evaluation

In multi-objective optimization, measuring the quality of a Pareto front approximation retrieved by an algorithm is not trivial. According to [2], a good approximation must minimize the distance to the actual Pareto front (a.k.a. convergence). Besides, a good Pareto front should consist of a highly diversified set of points, which are also well distributed across the front (a.k.a. spread and

distribution). Accordingly, we designed an evaluation to answer the following question: **Q1** How good are the Pareto fronts discovered by our proposal with respect to convergence, spread, and distribution? Secondly, as one of the goals of this paper is reducing the searching space through heuristics, our evaluation also answers the question: **Q2** How many solutions (objective function evaluations) do the algorithms need to explore to retrieve the Pareto front?

5.1 Implementation and Experimental Setup

To assess our proposal, we implemented the full approach presented in this paper in Python 3.8. Also, we adapted the resource allocation problem to the tool Pymoo [3], which implements the genetic algorithm NSGA-II. The source code and the instructions to execute the three variants, i.e., HC-STRICT, HC-FLEX, and TS-STRICT, and the NSGA-II algorithm, can be accessed from https://github.com/orlenyslp/bpm-r-opt.

In our experiments, we run the four algorithms HC-STRICT, HC-FLEX, TS-STRICT, and NSGA-II, taking the NSGA-II as a baseline to be compared with the results obtained by our approach. In the case of our approach, we set the maximum number of solutions to explore (i.e., function evaluations) to 10 000 and at most 800 (8%) consecutive Pareto non-optimal allocations. As for the NSGA-II, we configured the input with the default values recommended in [3], with a population size of 40 and a maximum of 250 generations (i.e., at most 10 000 function evaluations). For all the algorithms, we run 15 simulations per allocation (using the BIMP engine [1]) to calculate the values of $aCost$ and $cTime$. Also, to avoid giving any unfair advantages to an algorithm due to the simulations' stochastic nature, we memorized in files the simulation results. So, we can assert that if two algorithms explore the same resource allocation, they will get the same values of $aCost$ and $cTime$. Additionally, the memorization reduces the number of simulations, thus the execution times, when multiple algorithms explore common areas in the solution space.

As a starting point, we used simulation models derived from event logs using the Simod simulation discovery tool [5]. We derived simulation models from one synthetic event log and seven real-life ones. The synthetic log (namely *purchasing-example*) is part of the academic material of the Fluxicon Disco tool[3]. The first real-life log (*production*) is a log of a manufacturing process[4]. The second one (*consulta-data-mining*) is an anonymized log of an academic recognition process executed at a Colombian University, available in the Simod tool distribution. The third real-life log is a subset of the BPIC2012 log[5] – a log of a loan application process from a Dutch financial institution. We focused on the subset of this log consisting of activities that have both start and end timestamps. Similarly, we used the equivalent subset of the BPIC2017[6], which is an updated

[3] https://fluxicon.com/academic/material/.
[4] https://doi.org/10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399.
[5] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
[6] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

version of the BPIC2012 log (extracted in 2017 instead of 2012). We extracted the subsets of the BPIC2012 and BPIC2017 logs by following the recommendations provided by the winning teams of the BPIC 2017 challenge.[7]

Table 1. Characteristics of the business processes used in the experimentation.

	Production	Purchasing Example	Consulta Data-mining	Insurance	Call-centre	bpi-12	bpi-17-filt	bpi-17
Real traces	225	608	954	1182	3885	8616	8941	30 276
Simulation traces	550	1500	4000	4000	8000	18 000	18 000	24 000
Activities	23	23	18	11	8	8	9	9
Number of pools	7	6	4	2	3	4	2	3
Total resources	54	47	337	125	66	68	116	141
Simulation time	0.48	0.50	0.49	0.52	0.75	0.67	0.69	0.88

Table 1 gives descriptive statistics of the processes used in the experiments, such as the number of traces in the event log, the number of activities, resource pools, and the sum of the resources across the pools discovered by Simod. The row simulation time shows the average execution times (in seconds) obtained by running one simulation of the corresponding process using the BIMP engine. The number of traces produced per simulation (number of simulated traces) was set to at least two times the number of real traces to minimize stochastic variations.

Since data about salaries/costs of the resources involved in the process execution is missing in the event logs, we assigned each resource with the unitary cost for the experiments. Thus, the total resource pool cost is determined by the number of resources multiplied by the duration of the process execution, i.e., from the beginning of the first trace to the end of the last one.

5.2 Metrics and Experimental Results

As the actual Pareto front is unknown, we follow the approach presented in [7] which creates a reference Pareto front $PRef$ to compare the results retrieved by many solvers. Specifically, $PRef$ is the set containing the non-dominated (i.e., Pareto-optimal) solutions from the entire search space explored by all the runs of the four algorithms discussed in this paper. Henceforth, we will call $PRef$ the reference Pareto front (joint from many algorithms) and $PAprox$ the approximated (by one algorithm) Pareto front. Then, to answer the experimental question **Q1**, we used four metrics:

– Hyperarea [21] (HA) measures convergence and distribution. So far, it is considered the most relevant and widely used measure to compare algorithms in the evolutionary community [2]. Hyperarea is the area in the objective space dominated by a Pareto front delimited by a point $(c, t) \in \mathbb{R}^2$, which we set as the maximum cost and time among all the solutions explored. If $PRef$

[7] https://www.win.tue.nl/bpi/doku.php?id=2017:challenge.

Table 2. Results of the performance metrics.

		Insurance	bpi-17-filt	bpi-17	Call-centre	bpi-12	Consulta	Purchasing	Production
Hyperarea	HC-Strict	0.999715	0.999993	0.999388	0.979741	0.999999	0.939581	0.999966	0.984928
	HC-Flex	0.999878	0.999993	0.999702	0.979741	1.0	0.970996	0.999993	0.999878
	TS-Strict	0.999997	1.0	0.999989	0.999997	1.0	0.95116	0.999999	0.999878
	NSGA-II	1.0	1.0	0.999776	0.975051	1.0	0.999964	0.999995	0.991984
Hausdorff	HC-Strict	792 368.4	21 283.4	767 578.4	333 931.8	67 756.9	114 721.2	12 093.5	60 122.4
	HC-Flex	747 515.1	17 290.5	24 078.9	333 931.8	88 807.0	188 411.5	10 503.3	6102.8
	TS-Strict	1 307 871.7	0.0	23 275.6	25 645.8	55 421.4	209 925.0	12 411.9	6102.8
	NSGA-II	0.0	9033.2	42 653.2	426 892.2	105 021.1	11 404.6	93 709.6	775 721.6
Delta(Δ)	HC-Strict	1.247804	1.311345	1.314798	1.075864	1.128014	1.200819	1.471674	0.918048
	HC-Flex	1.119086	1.27965	0.856219	1.075864	1.181983	0.887497	1.454444	0.585820
	TS-Strict	1.117958	1.206904	0.876539	1.069856	1.197563	0.905387	1.496234	0.585820
	NSGA-II	1.458937	1.244832	0.753908	1.166828	0.995971	1.223205	1.326563	1.102538
Purity	HC-Strict	0.888889	0.90625	0.833333	0.333333	0.692308	0.777778	0.695652	0.466667
	HC-Flex	0.615385	0.911765	0.842105	0.333333	0.75	1.0	0.666667	0.923077
	TS-Strict	1.0	1.0	1.0	0.973684	0.666667	1.0	0.954545	0.923077
	NSGA-II	1.0	1.0	1.0	0.0625	1.0	1.0	0.848485	0.722222

is available, the hyperarea ratio is a real number, between 0 and 1, given by $HA(PAprox)/HA(PRef)$. A higher hyperarea ratio means a better $PAprox$, being 1 the maximum possible ratio indicating that $PAprox$ dominates the same solution space as $PRef$.

- Averaged Hausdorff distance [17] measures convergence using the distances between $PAprox$ and $PRef$. Specifically, it gets the greatest distance from each point in one set to the closest point in the other set, i.e., given by $max(min||p_i, PRef||_2, min||p_j, PAprox||_2)$, $\forall p_i \in PAprox, p_j \in PRef$. A lower Hausdorff distance means a better $PAprox$.
- Delta(Δ) [7,8] measures spread and distribution. It is given by the formula:

$$\Delta = \frac{d_0 + d_n + \sum_{i=1}^{n-1} |d_i - d'|}{d_0 + d_n + (n-1)d'}$$

where $d_i, 0 \leq i \leq n = |PAprox|$ is the Euclidean distance between consecutive solutions, with d_0 and d_n being the Euclidean distances between the extreme solutions in $PRef$ and the extreme solutions in $PAprox$. Besides, d' is the average of those distances. A lower value of Δ means a better $PAprox$.

- Purity [7] is a cardinality measure used to compare Pareto fronts constructed by different algorithms. It is given by $|PAprox \cap PRef|/|PAprox|$. Thus, it measures the ratio of solutions in $PAprox$ included in $PRef$. A higher purity means a better $PAprox$ in terms of percentage of non-dominated solutions, being 1 the maximum value possible.

Table 2 shows the results of the performance metrics achieved by the four algorithms, highlighting the best score for each metric on each of the event logs.

The experiments show that, in most of the logs, the tabu-search TS-STRICT scored the best results in each of the four metrics assessed, followed by the genetic algorithm NSGA-II. As expected, the algorithm hill-climbing HC-STRICT exposes the lowest performance among all the solvers, with its flexible variant HC-FLEX, i.e., considering the MAD deviation, improving its results. However, both

variants of hill-climbing also constitute good initial approximations of the Pareto front. They exhibit performances that are close in terms of the metric evaluations to the NSGA-II and TS-STRICT. In all the cases, the algorithms scored hyper-area ratios superior to 0.93 (being 1.0 the max possible), meaning that they dominate at least 93% of the solution space dominated by the reference Pareto front. Also, the Hausdorff distances and Δ spread do not evidence a bad performance of any algorithm compared to the others. For example, although HC-STRICT never obtains the best measurement, it achieves better scores than the NSGA-II algorithm in 50% of the logs in both metrics. Finally, the purity rates show that both variants of hill-climbing add fewer points to $PRef$ compared to TS-STRICT and NSGA-II. These results were expected as hill-climbing uses a more local searching strategy. Thus, it explores a reduced number of allocations but still discovers Pareto fronts with sound values of convergence, spread, and distribution according to the Hyperarea, Hausdorff, and Δ metrics.

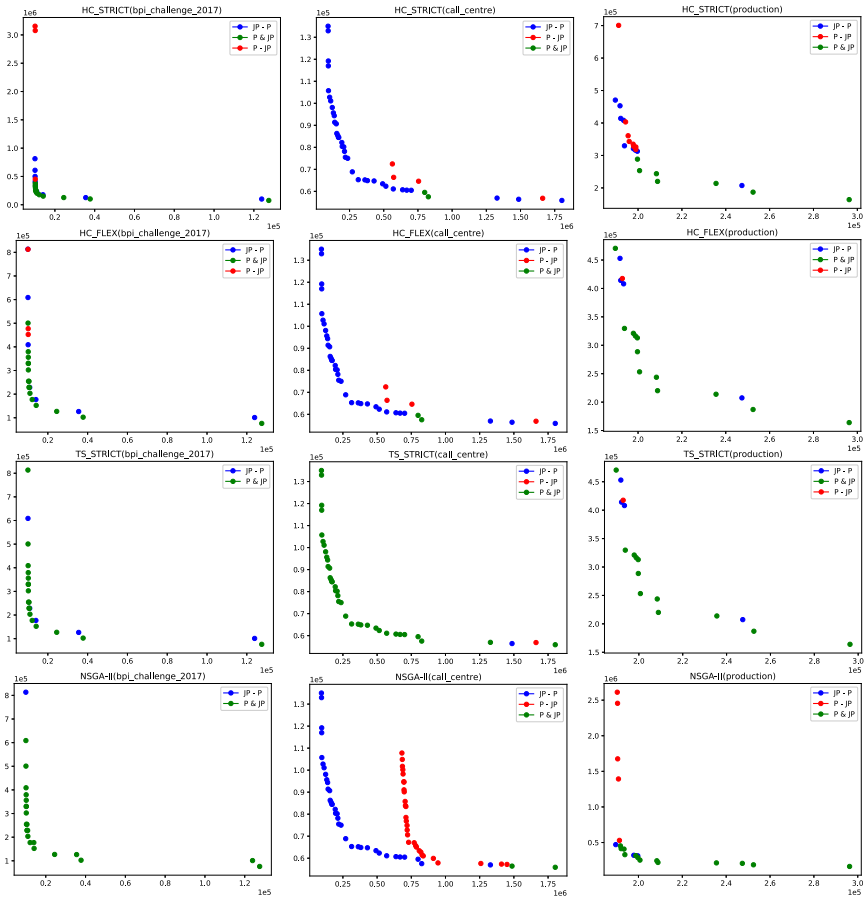

Fig. 1. Approximated Pareto Fronts from logs *bpi-17*, *call-centre* and *production*. The times in *cTime* (y-axis) and *procDuration* to compute *aCost* (x-axis) are in seconds.

Figure 1 compares the Pareto fronts $PAprox$ discovered by each algorithm from the event logs *bpi-17*, *call-centre* and *production* with the reference Pareto front $PRef$.[8] The points ($aCost$, $cTime$) in blue are in $PRef$ but not in $PAprox$, the ones in green are in $PRef$ and $PAprox$, while the ones in red are the points in $PAprox$ but not in $PRef$. The figure illustrates how the approximations obtained by TS-STRICT and NSGA-II share more solutions with $PRef$, while the solutions discovered by the hill-climbing variants span local regions of $PRef$. Across all the experiments, the event log *call-centre* (in the middle) led to the most disperse results. In this case, TS-STRICT found 37 of the 38 non-dominated points in $PRef$, while NSGA-II, HC-STRICT, and HC-FLEX found different sets containing only 2 of the points in $PRef$. However, those points excluded from $PRef$ were still close, dominating a high solution space region as the hyper-area ratio shows.

To answer the research question **Q2**, we use data profiles [15] to assess how well each algorithm performed in terms of number of evaluations of the objective function (that require the calculation of $aCost$ and $cTime$). Specifically, we plot the cumulative percentage of non-dominated solutions added in $PAprox$ after a given number of function evaluations.

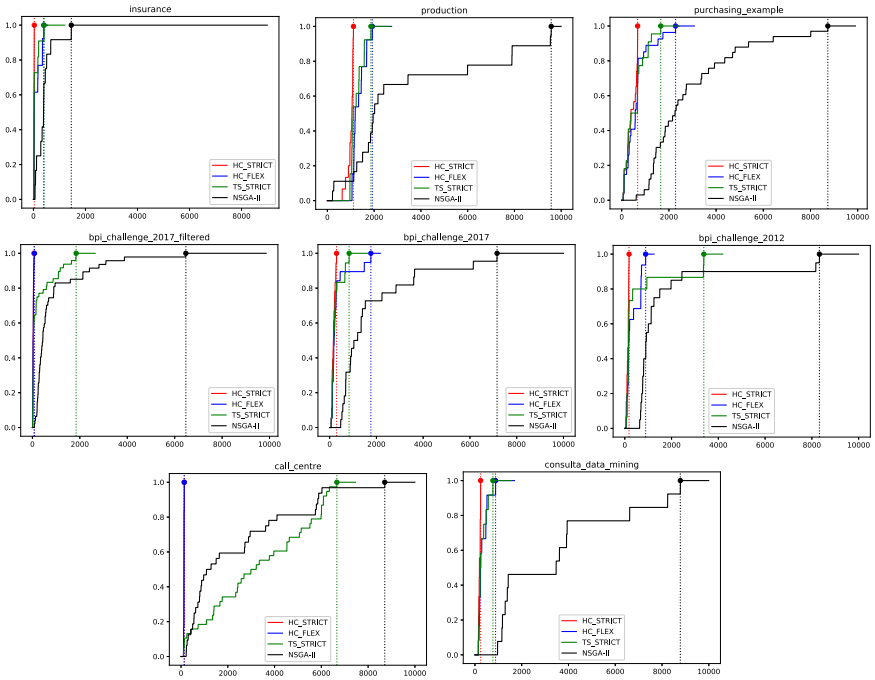

Fig. 2. Pareto front growing ratio (y-axis) in terms of function evaluations (x-axis).

[8] The Pareto fronts from the remaining event logs, and the full results obtained in the experiments, can be accessed from the code repository.

Figure 2 illustrates the cumulative growing ratio of the Pareto fronts $PAprox$, i.e., between 0 (no points) and 1 (all the points in $PAprox$), progressing with the number of function evaluations. The figure sketches with the dotted lines at which function evaluation each algorithm added the last point in $PAprox$. As expected, hill-climbing achieves the best performance, carrying out a significantly lower number of function evaluations than the other solvers. The algorithm NSGA-II shows the worst performance, followed by TS-STRICT. Unlike the variants proposed in this paper that explore each resource allocation only once, the NSGA-II algorithm may explore a resource allocation several times, thus requiring multiple calculations of $aCost$ and $cTime$. Regarding the different resource allocations explored, on average, the HC-STRICT traversed 361 allocations, HC-FLEX 1468, TS-STRICT 2982, and NSGA-II 4641.

The experimental evaluation evidenced that HC-STRICT, followed by HC-FLEX, requires fewer function evaluations to construct a Pareto front with acceptable accuracy. However, they also discover fewer non-dominated solutions than TS-STRICT, which exhibits Pareto fronts with higher accuracy. Accordingly, we can conclude that the hill-climbing variants provide a proper initial approximation of the Pareto fronts so that the business analysts obtain a solution in shorter computational times. In contrast, the TS-STRICT (like NSGA-II) provides more accurate and varied Pareto optimal allocations so that analysts have a broader range of choices but at the cost of exploring more solutions.

6 Conclusion

This paper presented an optimization approach to compute a set of Pareto-optimal resource allocations minimizing the cost and cycle time of a process. The approach heuristically explores the search space of possible resource allocations using a simulation model to evaluate each allocation. The approach incorporates a perturbation method that selects solution candidates that are likely to Pareto-dominate the already explored allocations based on two indicators: resource utilization and resource impact. The approach also incorporates a ranking method that sorts the resource allocations, exploring the closest one to the current Pareto front so as to accelerate the convergence. The perturbation and ranking methods are embedded into two variants of the hill-climbing meta-heuristic, namely HC-STRICT and HC-FLEX, and a variant of tabu-search, namely TS-STRICT. In HC-FLEX, we relax the definition of Pareto-domination so as to prevent that the algorithm is trapped too quickly into a local optimum due to stochastic variations in the outputs of the simulation model.

The experimental evaluation shows that our approach requires fewer evaluations of the objective functions to retrieve Pareto fronts of quality comparable to those discovered by the NSGA-II algorithm. Moreover, with sufficient iterations, the tabu-search approach leads to higher-quality Pareto fronts than NSGA-II.

A limitation of the current approach is that the exploration of the search space is done in a sequential manner. An avenue for future work is to speed up the approach by parallelizing the generation of solution candidates and their evaluation via simulation. Secondly, our proposal focuses on optimizing the number

of resources per pool, assuming that all resources in a pool have identical characteristics. Another future work direction is to extend the approach to models with differentiated resources (e.g., different resources have different performance) as well as resources shared across pools or processes.

Acknowledgment. Work funded by European Research Council (PIX project).

References

1. Abel, M.: Lightning Fast Business Process Model Simulator. Master's thesis, University of Tartu (2011)
2. Audet, C., Bigeon, J., Cartier, D., Le Digabel, S., Salomon, L.: Performance indicators in multiobjective optimization. Eur. J. Oper. Res. **292**(2), 397–422 (2021)
3. Blank, J., Deb, K.: Pymoo: multi-objective optimization in python. IEEE Access **8**, 89497–89509 (2020)
4. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Inf. Sci. **237**, 82–117 (2013)
5. Camargo, M., Dumas, M., González, O.: Automated discovery of business process simulation models from event logs. Decis. Support Syst. **134**, 113284 (2020)
6. Chaharsooghi, S.K., Kermani, A.H.M.: An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP). Appl. Math. Comput. **200**(1), 167–177 (2008)
7. Custódio, A.L., Madeira, J.F.A., Vaz, A.I.F., Vicente, L.N.: Direct multisearch for multiobjective optimization. SIAM J. Optim. **21**(3), 1109–1140 (2011)
8. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
9. Djedović, A., Žunić, E., Avdagić, Z., Karabegović, A.: Optimization of business processes by automatic reallocation of resources using the genetic algorithm. In: IEEE BIHTEL 2016 Proceedings, pp. 1–7 (2016)
10. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56509-4
11. Durán, F., Rocha, C., Salaün, G.: Analysis of resource allocation of BPMN processes. In: ICSOC 2019 Proceedings, pp. 452–457 (2019)
12. Huang, Z., van der Aalst, W.M.P., Lu, X., Duan, H.: Reinforcement learning based resource allocation in business process management. Data Knowl. Eng. **70**(1), 127–145 (2011)
13. Huang, Z., Lu, X., Duan, H.: A task operation model for resource allocation optimization in business process management. IEEE Trans. Syst. Man Cybern. Part A **42**(5), 1256–1270 (2012)
14. Lee, H., Kim, S.S.: Integration of process planning and scheduling using simulation based genetic algorithms. Int. J. Adv. Manuf. Technol. **18**(8), 586–590 (2001)
15. Moré, J.J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. SIAM J. Optim. **20**(1), 172–191 (2009)
16. Peters, S.P.F.: Analysis and Optimization of Resources in Business Processes. PhD dissertation, Technische Universiteit Eindhoven (2021)
17. Schütze, O., Esquivel, X., Lara, A., Coello, C.A.C.: Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization. IEEE Trans. Evol. Comput. **16**(4), 504–522 (2012)

18. Senkul, P., Toroslu, I.H.: An architecture for workflow scheduling under resource allocation constraints. Inf. Syst. **30**(5), 399–422 (2005)
19. Si, Y., Chan, V., Dumas, M., Zhang, D.: A Petri nets based generic genetic algorithm framework for resource optimization in business processes. Simul. Model. Pract. Theory **86**, 72–101 (2018)
20. Weise, T.: Global optimization algorithms-theory and application. Self-Published Thomas Weise (2009)
21. Wu, J., Azarm, S.: Metrics for quality assessment of a multiobjective design optimization solution set. J. Mech. Des. **123**(1), 18–25 (2001)
22. Xu, J., Liu, C., Zhao, X., Yongchareon, S., Ding, Z.: Resource management for business process scheduling in the presence of availability constraints. ACM Trans. Manag. Inf. Syst. **7**(3), 9:1–9:26 (2016)
23. Yu, Y., Pan, M., Li, X., Jiang, H.: Tabu search heuristics for workflow resource allocation simulation optimization. Concurr. Comput. Pract. Exp. **23**(16), 2020–2033 (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.