An Algebra of Pareto Points*

Marc Geilen, Twan Basten, Bart Theelen and Ralph Otten
Eindhoven University of Technology
{m.c.w.geilen,a.a.basten,b.d.theelen,r.h.j.m.otten}@tue.nl

Abstract

Multi-criteria optimisation problems occur naturally in engineering practices. Pareto analysis has proven to be a powerful tool to characterise potentially interesting realisations of a particular engineering problem for design-space exploration. Depending on the optimisation goals, one of the Pareto-optimal alternatives is the optimal realisation. It occurs however, that partial design decisions have to be taken, leaving other aspects of the optimisation problem to be decided at a later stage, and that Pareto-optimal configurations have to be composed (dynamically) from Pareto-optimal configurations of components. Both aspects are not supported by current analysis methods. This paper introduces a novel, algebraic approach to Pareto analysis. It allows for describing incremental design decisions and composing sets of Pareto-optimal configurations. The algebra can be used to study the operations on Pareto sets and the efficient computation of Pareto sets and their compositions.

1 Introduction

In multi-criteria optimisation or decision problems there are multiple, potentially dependent and conflicting objectives. Depending on the relative importance of the objectives, different solutions may be preferred. The notion of Pareto optimality, named after Vilfredo Pareto who introduced it in [11] as '*maximum ophelimity*' in an economic system, states that a solution is optimal if it is impossible to find a solution which improves on one or more of the objectives without worsening any of them. If one solution is better in one objective than another solution and not worse in any other objectives, the latter is *dominated* by the former, which should always be preferred. Exploring the possible solutions to an optimisation problem then amounts to finding all solutions that are not dominated by any other (feasible) solution. This set of solutions is called the Pareto frontier and is guaranteed to contain all optimal solutions, whatever way the individual objectives are weighted. In other words, the Pareto frontier exactly captures the available trade-offs between the different objectives.

Multi-criteria optimisation problems appear frequently in a wide range of fields including business, economics and all kinds of engineering activities. In this paper, we briefly discuss some engineering applications, and in particular dynamic Quality-of-Service management in wireless networks of embedded resource-constrained devices. Selecting an optimal configuration for a dynamic constellation of devices and available resources requires optimisation of power consumption, quality, timeliness, and so forth, at run-time. The available configurations of the components themselves may be obtained from off-line Pareto analysis, of different architecture mappings for instance. Some configurations may be good for efficient use of energy, others for obtaining a higher quality, and so forth. The selection of a configuration at run-time requires composition of Pareto-optimal configurations of parts into Pareto-optimal configurations of a system. Some of these operations will have to be performed at run-time on resource-constrained devices. The number of configurations will have to be kept small and partial selection among optimal configurations may be needed to keep the number relatively small.

This paper introduces an algebraic approach to Pareto-optimal sets of configurations and defines operations on such sets. The algebra of sets of Pareto points provides a formal framework that allows to study compositions of Pareto sets and their properties, to define how to compute them and to apply cost functions to select optimal configurations or reduce the number of configurations to be stored.

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 gives motivation for the algebra introduced in this paper and introduces an example that is used throughout the paper to exemplify the concepts that are presented. Section 4 formalises the well-known notions of Pareto dominance and Pareto optimality. In Section 5, we take a closer look at cost functions that provide a ranking of Pareto points, their relationship with Pareto analysis and their use in intermediate, partial selection to reduce the size of Pareto sets. Section 6 then introduces the operators of the algebra and studies their properties, in particular w.r.t. Pareto optimality of the operands and the composition results. For practical use of the algebra to compute sets of configurations, we look at common combinations of the operators in Section 7. Section 8 discusses computational and complexity issues and Section 9 concludes. Proofs have been omitted because of space limitations. An extended version of the paper with proofs can be found in [6].

*This work is supported by the IST - 004042 project, Betsy.

2 Related Work

Pareto analysis is used in many engineering practices [16]. In our own field of research for instance, we can mention its use in design-space exploration [9, 14], application mapping on embedded multiprocessor systems [13, 15] or architecture exploration for Systems-on-Chip (SoCs) [7]. There is also a vast body of work on Pareto analysis and multi-objective optimisation and decision making in general (see, e.g., [4] for an overview). This kind of work traditionally focusses on characterisation of the Pareto frontier based on the nature of the cost (objective) functions (typically continuous functions on real numbers), algorithms for computing the Pareto frontier or approximations of the frontier [10, 18]. We do not look at particular objectives or cost functions, but rather look at the algebraic structure and properties, requiring only that an objective or cost function be monotone (which we show in this paper to be necessarily so for the Pareto approach to be adequate). The particular novelty of our approach lies in the fact that we pursue an incremental, compositional approach to determining Pareto sets, based on a rigourous algebra of well-defined operations on Pareto sets. A new aspect is the use of partially ordered quantities as the parameters of solutions, which turns out to be the crucial point to obtain the compositionality for an algebra. To the best of our knowledge, this paper is the first to take an algebraic approach to Pareto points.

Algorithms to compute the Pareto frontier include genetic or evolutionary algorithms [3, 17, 18], and tabu search algorithms [1, 8]. Algorithms to identify Pareto points in sets of configurations (also called Pareto minimisation) are introduced, although according to [16], literature on identifying Pareto sets is sparse. The algorithm of Bentley [2] is asymptotically very efficient. [16] introduces a hybrid algorithm which combines it with a straightforward algorithm which is efficient for small sets. Data structures for storing sets of configurations have also been studied [12, 16]. In particular quad-trees have been introduced to manipulate such sets more efficiently, especially when the sets are large and the number of objectives is large. The results of these efforts are also useful for implementing the algebraic operations in this paper and reduction of the sets of configurations by approximation when their sizes become too large.

3 Motivating Examples

To further motivate our algebra of Pareto points, we elaborate a bit on two potential applications in electronic system design. The two examples are of a complementary nature, each with their own specific requirements.

3.1 Design Closure in SoC Design

Design closure for electronic design automation (EDA) tools means that users can specify a function to be implemented on a chip, feed it to an EDA tool, and get, without further interaction, a design that meets all requirements concerning functionality, speed, size, power, yield, and other "costs". The EDA industry has thus-far concentrated on tools and techniques to achieve closure with respect to individual target design parameters at the lower levels of abstraction. Today's chip synthesis requires tools and methodologies for manipulating designs at higher levels of abstraction to meet a large variety of performance constraints. The algebra of this paper can support trade-offs between performance characteristics over many levels of abstraction.

Around 1980, designers realised that the complexity of chips would force them to use more than just a routing and occasionally a placement algorithm to find a starting point. It was obvious that the two tasks were heavily dependent on each other. Routing without a placement was inconceivable while at the same time a placement might render any routing infeasible. A solution to this dependency problem, named *wiring closure*, was found with a generalisation of placement called floor-plan design, slicing and floor-plan optimisation, resulting in a trade-off between the chip dimensions without design iterations. Later, a similar thing happened with the *timing closure* problem in which a trade-off between size and speed of gates is obtained. After timing, the primary concern for today's *SoC* devices is power consumption. The trade-off is between speed and energy efficiency of circuits.

In current practice, closure is achieved by Pareto style trade-offs. Every realisation has a certain combination of performance characteristics. The best Pareto point for a specific cost function is selected. Premature commitment to a particular realisation while unaware of consequences later in the design trajectory, implies the risk that the realisation turns out to be not the optimal one. Nowadays, there are many relevant characteristics, interacting with different levels of the design process. Choosing the 'best' one at a certain level may preclude optimal ones at later stages. Multidimensional Pareto analysis can help to solve these problems by its ability to partially resolve design decisions at a certain stage if it can be done safely, and use the results, compositionally, at later stages in the design. This way realisation options are not unnecessarily lost.

This example illustrates a typical off-line, design-time analysis scenario, which may involve large numbers of Pareto points and being off-line, the time needed for minimisation is not so important. In contrast, for run-time analysis, the time for minimisation is crucial and the number of Pareto points that need to be manipulated has to be limited. On-line analysis may build upon partial optimisation results determined off-line. Such an example is presented next.

3.2 Run-time reconfiguration and QoS

To optimise Quality-of-Service and Quality-of-Experience, modern multimedia and consumer electronics systems need to adapt to dynamically changing circum-

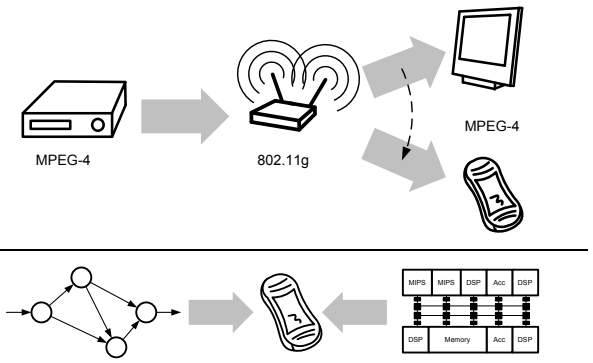


Figure 1. Wireless follow-me scenario

stances and needs or desires of users. An ad-hoc formation of components needs to be configured dynamically by a QoS management system. To allow QoS management to use a component to its full extent, the component needs to expose its configuration options and trade-offs to the manager. The optimal working configuration is partly determined off-line by analysing possible realisations, and partly on-line by combining the trade-offs of all components to select a globally optimal configuration, given decision parameters such as user preferences. A component may expose its configuration options to the QoS manager in the form of a Pareto set.

When Pareto sets are used at run-time to select working configurations, the representation of the trade-offs must be very efficient and the run-time computation of the system trade-offs from component trade-offs and the run-time analysis of the optimal configuration need to be very fast.

Figure 1 shows a simple application scenario in this context, which is used throughout the paper to illustrate the concepts introduced, and their practical application. A server connected to a wired infrastructure encodes a video stream for transmission over a wireless network to a large television screen. We consider the moment that the user moves away from the screen and the system initiates a reconfiguration that moves the stream from the large screen to a hand-held device that the user can take with him/her. The display devices decode the stream and display it on the screen. The hand-held itself consists of a hardware platform with different kinds of processing elements. The decoder and display applications are mapped onto this hardware platform and different mappings result in different characteristics (see also Figure 1). The following QoS-related trade-offs and properties play a role for this system.

- The server has a possibility to choose different encodings of the stream. The quality and required bandwidth of the generated stream depend on this.
- The wireless network imposes a constraint on the available bandwidth, this bandwidth may fluctuate.
- An application on the hand-held has a trade-off between quality, the bandwidth of the stream (essentially the same trade-off as in the server) and computational effort required from the platform for the decoding.
- The platform of the hand-held is able to deliver different levels of computational power at different costs in terms of energy consumption.

The properties of the application-to-platform mapping in the hand-held can be analysed off-line and the resulting options can be encoded into operating modes of the device. At run-time, when a user uses the device for displaying a video stream in a particular setting, a QoS management system can select the most appropriate mode of the hand-held and all other system components. When conditions change, e.g. a drop in bandwidth or a low battery, the QoS manager may initiate a reconfiguration procedure to adapt the working mode to the new circumstances.

4 Pareto Points

Pareto points describe the different qualities of possible system configurations, such as the bandwidth used, time needed for completion, energy consumption, etc. We call these different dimensions of a configuration *quantities*. Quantities can be ordered in terms of better and worse, although we allow such an ordering to be partial. The latter allows us to consider, for example, the composition of execution time and energy consumption as a quantity.

Definition 1 (QUANTITY) *A quantity is a set Q with a partial order $\preceq_Q$. If $\preceq_Q$ is total, the quantity is called* basic. *If the quantity is clear from the context we denote the order just by $\preceq$.*

Smaller quantities are preferred over larger ones. The quality level of the video encoding in the server of our example is, for instance, characterised with the quantity

$$SQuality = \{sq\_high, sq\_med, sq\_low\}$$
$$sq\_high \preceq sq\_med \preceq sq\_low.$$

Quality settings are only ordered relative to each other; being the result of human perception, it is inherently hard to quantify quality by numbers. The corresponding server bandwidth requirements are captured with the following quantity.

$$SBandwidth = \{sb\_high, sb\_med, sb\_low\}$$
$$sb\_low \preceq sb\_med \preceq sb\_high.$$

These values can be quantified, e.g., by profiling the streams. For readability however, we do not use concrete numbers here. For available network bandwidth, we use the quantity

$$NBandwidth = \{nw\_low, nw\_med, nw\_high\}$$
$$nw\_high \preceq nw\_med \preceq nw\_low.$$

Note the difference with the server bandwidth, where a low bandwidth is preferred; for the network, a higher available bandwidth is better. The video encoding modes are captured in the unordered, i.e., partially ordered, quantity

$$EncodingMode = \{A, B, C\} \text{ with } \preceq = \varnothing$$

Encoding modes cannot freely be determined by the server itself. Preferences among modes may originate from derived properties such as quality or available bandwidth.

System configurations are selected from a *configuration space* built up from quantities.

Definition 2 (CONFIGURATION SPACE) *A configuration space $\mathcal{S}$ is the Cartesian product $Q_1 \times Q_2 \times \ldots \times Q_n$ of a finite number of quantities.*

Definition 3 (CONFIGURATION) *A configuration $\bar{c} = (c_1, c_2, \ldots, c_n)$ is an element of configuration space $Q_1 \times Q_2 \times \ldots \times Q_n$. We use $\bar{c}(Q_k)$ or $\bar{c}(k)$ to denote c_k.*

Sets $\mathcal{C} \subseteq \mathcal{S}$ of configurations are used to represent the different options for realising a particular system or component. The configuration space of the server in the example could for instance be $EncodingMode \times SQuality \times SBandwidth$. The configurations could e.g. be

$$\{(A, sq\_low, sb\_low), (A, sq\_low, sb\_med),$$
$$(B, sq\_med, sb\_med), (C, sq\_high, sb\_high)\}$$

which could be determined by profiling different realisations of the application.

A partial ordering of configurations is induced from the order on the individual quantities by a point-wise ordering.

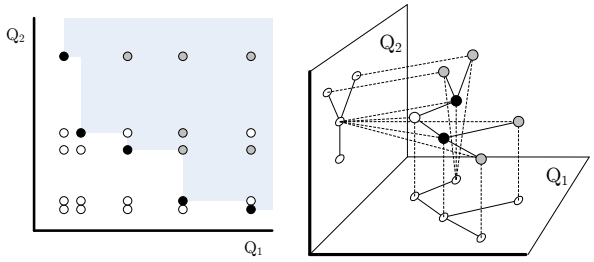
Definition 4 (DOMINANCE) *If $\bar{c}_1, \bar{c}_2 \in \mathcal{S}$, then $\bar{c}_1 \preceq \bar{c}_2$ iff for every quantity Q_k of $\mathcal{S}$, $\bar{c}_1(Q_k) \preceq_{Q_k} \bar{c}_2(Q_k)$. If $\bar{c}_1 \preceq \bar{c}_2$, then $\bar{c}_1$ is said to* dominate *$\bar{c}_2$.*

Dominance of one configuration over another thus expresses the fact that the configuration is at least as good, because it is at least as good in every aspect. Note that a configuration space with the dominance ordering is itself a quantity again, a set with a partial order. For convenience, dominance is reflexive, i.e., a configuration dominates itself. The irreflexive, *strict dominance*, is denoted as $\prec$. $(A, sq\_low, sb\_low) \prec (A, sq\_low, sb\_med)$ because the former provides the same quality at a lower bandwidth for the same mode. A configuration that is strictly dominated by another one is usually not an interesting one, so we would like to remove it from a set of configurations. Sets of configurations that cannot be reduced without sacrificing potentially interesting realisations are called Pareto minimal (sometimes also Pareto optimal or efficient, or Pareto set).

Definition 5 (PARETO MINIMAL) *A set $\mathcal{C}$ of configurations is said to be* Pareto minimal *iff for any $\bar{c}_1, \bar{c}_2 \in \mathcal{C}$, $\bar{c}_1 \not\prec \bar{c}_2$ (an anti-chain of the dominance relation).*

Minimality states that the set does not contain any strictly dominated configurations. Individual elements of a Pareto minimal set of configurations are referred to as *Pareto points*, the set itself is often called the *Pareto frontier*. If we remove the dominated configuration from the configurations of the server, we obtain a Pareto minimal set:

$$\{(A, sq\_low, sb\_low), (B, sq\_med, sb\_med),$$
$$(C, sq\_high, sb\_high)\}$$



(a) (b)

Figure 2. Compositionality using Pareto sets with partially ordered quantities

To compute Pareto minimal sets, we need to modify sets of configurations. The following definitions formalise that no interesting solutions are lost in such a process. First, the dominance of configurations is lifted to configuration sets.

Definition 6 (SET DOMINANCE) *A set $\mathcal{C}_1$ of configurations from configuration space $\mathcal{S}$ dominates a set $\mathcal{C}_2$ of configurations of $\mathcal{S}$, denoted as $\mathcal{C}_1 \preceq \mathcal{C}_2$, iff for every $\bar{c}_2 \in \mathcal{C}_2$ there is some $\bar{c}_1 \in \mathcal{C}_1$ such that $\bar{c}_1 \preceq \bar{c}_2$.*

We say that two sets of configurations are equally good if they dominate each other.

Definition 7 (PARETO EQUIVALENCE) *Two configuration sets $\mathcal{C}_1$ and $\mathcal{C}_2$ are Pareto equivalent, denoted $\mathcal{C}_1 \equiv \mathcal{C}_2$, iff $\mathcal{C}_1 \preceq \mathcal{C}_2$ and $\mathcal{C}_2 \preceq \mathcal{C}_1$.*

Pareto equivalence essentially claims that neither of the sets contains a configuration that cannot be matched or improved upon by the other, or in other words, that they have the same Pareto points. It is easy to prove that $\equiv$ is indeed an equivalence relation. This notion of equivalence is often left implicit in the discussion of Pareto analysis. For our algebra however it is useful to make it precise. A Pareto minimal set of configurations can now also be characterised as a set from which no configuration can be removed with an equivalent set as a result. Note that $\mathcal{C}_1$ and $\mathcal{C}_2$ need not be the same to be equivalent. However, equivalent sets can be reduced to the same Pareto minimal set while maintaining equivalence. In Section 6 we return to this point.

Example 1 *Figure 2(a) shows a configuration space of the basic quantities Q_1 and Q_2. The dots represent the configurations of the configuration space. The gray configurations are all dominated by at least one of the black configurations. The set of all black configurations is Pareto minimal and Pareto equivalent to the set of all black and gray configurations. If one has computed for instance the black set of configurations of a system, then the white configurations typically represent configurations that are infeasible (or not known to be feasible).*

Figure 2(a) shows a classical configuration space and Pareto points. A novel aspect of our approach is that it generalises quantities allowing them to be partially ordered. We tried to

illustrate this case in Figure 2(b). To not clutter the picture, only a few configurations are shown. Again, the black configurations dominate the gray ones, because they are identical in one configuration, and better in the other. The two black configurations are incomparable, because they are incomparable in quantity Q_1.

Equivalent Pareto minimal sets are necessarily the same.

Proposition 1 *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are two Pareto minimal sets of configurations and $\mathcal{C}_1 \equiv \mathcal{C}_2$, then $\mathcal{C}_1 = \mathcal{C}_2$.*

PROOF Let $\bar{c}_1 \in \mathcal{C}_1$. By dominance of $\mathcal{C}_2$ over $\mathcal{C}_1$, there is some $\bar{c}_2 \in \mathcal{C}_2$ such that $\bar{c}_2 \preceq \bar{c}_1$. Conversely, by dominance of $\mathcal{C}_1$ over $\mathcal{C}_2$, there is some $\bar{c}'_1 \in \mathcal{C}_1$ such that $\bar{c}'_1 \preceq \bar{c}_2 \preceq \bar{c}_1$. Since $\mathcal{C}_1$ is minimal, $\bar{c}'_1 = \bar{c}_2 = \bar{c}_1$ and $\bar{c}_1 \in \mathcal{C}_2$. Thus, we conclude that $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and repeating the proof, switching the roles of $\mathcal{C}_1$ and $\mathcal{C}_2$, it follows that also $\mathcal{C}_2 \subseteq \mathcal{C}_1$ and hence, $\mathcal{C}_1 = \mathcal{C}_2$. □

The following theorem shows that all well-ordered sets of configurations have a unique minimal equivalent, its Pareto frontier. A partial order $(D, \sqsubseteq)$ is called well-ordered if every chain contains a smallest element.

Theorem 1 *If $\mathcal{C}$ is a set of configurations and $(\mathcal{C}, \preceq)$ is well-ordered, then there is a unique Pareto minimal set $\mathcal{D}$ such that $\mathcal{D} \equiv \mathcal{C}$.*

PROOF We need to prove that (i) there exists an equivalent minimal set and (ii) it is unique. (i) for every configuration $\bar{c} \in \mathcal{C}$, we can pick a configuration $\bar{d}_{\bar{c}} \in \mathcal{C}$ that dominates $\bar{c}$, but is itself not strictly dominated by any configuration of $\mathcal{C}$, because $(\mathcal{C}, \preceq)$ is well-ordered. Let $\mathcal{D} = \{\bar{d}_{\bar{c}} \mid \bar{c} \in \mathcal{C}\}$. Then $\mathcal{D}$ is Pareto minimal and $\mathcal{D} \equiv \mathcal{C}$. (ii) Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be Pareto minimal sets of configurations and Pareto equivalent to $\mathcal{C}$. Then, since $\mathcal{D}_1 \equiv \mathcal{D}_2$, Proposition 1 shows that $\mathcal{D}_1 = \mathcal{D}_2$. □

E.g., the infinite set $\mathbb{N}$ of natural numbers has a Pareto minimal equivalent: $\{1\}$ and moreover, every set of natural numbers has a minimum (which is known as the Well Ordering Principle). The interval $(1, 2]$ of real numbers however, is not well-ordered. Such sets have to be avoided. As a direct consequence of the previous theorem and the fact that every finite partial order is well-ordered, we know that every finite set of configurations has a unique minimal equivalent.

Corollary 1 *For every finite set $\mathcal{C}$ of configurations, there is a unique Pareto minimal set $\mathcal{D}$ such that $\mathcal{D} \equiv \mathcal{C}$.*

In the remainder of this paper, we assume that sets of configurations are well-ordered.

5 Cost Functions

Pareto points represent potentially interesting system realisation alternatives. When the time comes to actually select one, single configuration to be realised, additional information is required to determine which configuration is best. This additional information is typically given as a cost function that can be applied to all the configurations and one with minimal cost is chosen.

5.1 Pareto Minimal Sets as Abstractions of Cost Functions

One way to look at Pareto points is as an abstraction of specific cost functions. With a clear objective, expressed as a cost function, an optimal configuration can be selected and alternative configurations are in principle irrelevant. One may choose to look at one single quantity, such as the energy consumption in the hand-held in our example, or the quality of the stream, but also at a weighted combination of both objectives. Sometimes however, one would like to resolve the decision only partially and let some independent objectives remain. In [13], for instance, a weighted sum of sequential and parallel execution times of an application is computed because the former is more relevant on a sequential platform, while the latter is more relevant on a parallel platform. The result is evaluated for different combinations of weights to obtain good solutions for both objectives. It would be more natural here to use a partially ordered cost function which retains parallel as well as sequential optimal execution times of solutions, without the need for weight factors. Our framework supports such incremental, partial decision making using partially ordered quantities.

An important observation is that the exact cost function is often unknown at a certain moment in the design. Pareto points represent exactly those configurations that can potentially be optimal under certain 'reasonable' cost functions. It does not make sense to take just any conceivable cost function into account, since then any configuration can potentially be optimal. A cost function is reasonable if it behaves monotonically w.r.t. the design's quantities. If a configuration dominates another configuration, then the cost should not get worse. Under this assumption, the Pareto points capture precisely the potentially interesting configurations. This is captured by the following two properties given below. Firstly, for a monotone cost function, a dominated configuration cannot have a better cost. (The Pareto points are sufficient.) Conversely, for any Pareto minimal set and a single configuration in this set, there exists a cost function for which that configuration has optimal cost. (All Pareto points are necessary.) Note that the domain of a cost function is simply a partially ordered set, i.e., a quantity.

Definition 8 (MONOTONICITY) *A function $f : \mathcal{S} \to Q$ from configuration space $\mathcal{S}$ to quantity Q is* monotone *iff for any $c_1, c_2 \in \mathcal{S}$ such that $c_1 \preceq c_2$, $f(c_1) \preceq_Q f(c_2)$.*

Definition 9 (COST FUNCTION, OPTIMALITY) *A* cost function *on a configuration space $\mathcal{S}$ is a monotone function $f : \mathcal{S} \to Q$. A configuration $\bar{c}$ is called the* optimum in the set $\mathcal{C} \subseteq \mathcal{S}$ of configurations for cost f *iff for any configuration $\bar{c}' \in \mathcal{C}$, $f(\bar{c}) \preceq f(\bar{c}')$ and $\bar{c}$ is called* optimal in $\mathcal{C}$ for f *iff for every configuration $\bar{c}' \in \mathcal{C}$, $f(\bar{c}') \not\prec f(\bar{c})$. Cost function f can be lifted to sets of configurations: $f(\mathcal{C}) = \{f(\bar{c}) \mid \bar{c} \in \mathcal{C}\}$.*

In contrast with traditional Pareto analysis approaches, we allow the cost function to return a value in a partially ordered domain. This means that application of the cost function adds information about the preference of Pareto points, but need not make a definitive selection of a single optimum configuration. Multiple configurations may be optimal and an optimum need not exist. This enables an incremental compositional way of computing trade-offs allowing for partial selection decisions to be taken at different places in the process. Note also that monotonicity of the cost function implies that the cost of two configurations with a dominance relationship can always be compared (the dominating configuration must have better cost). The following two theorems (which are variations of the two Fundamental Theorems of Welfare Economics [11]) prove that the Pareto minimal sets of configurations capture precisely the required information to guarantee that an optimal configuration can be selected for any arbitrary monotone cost function.

Theorem 2 *(*SUFFICIENCY OF PARETO SETS*) Let $f : \mathcal{S} \to Q$ be a cost function. If $\bar{c}_1, \bar{c}_2 \in \mathcal{S}$ with $\bar{c}_1 \preceq \bar{c}_2$, then $f(\bar{c}_1) \preceq f(\bar{c}_2)$.*

Theorem 3 *(*NECESSITY OF PARETO SETS*) Let $\mathcal{C}$ be a Pareto minimal set of configurations of $\mathcal{S}$ and $\bar{c} \in \mathcal{C}$. Then, there exists a cost function f on $\mathcal{S}$ such that $\bar{c}$ is optimal (the optimum) for f.*

5.2 Priority

An example of a practically useful cost function that may only partially resolve the choice of optimal configuration is prioritisation. The size of a set of configurations can be reduced by introducing a prioritisation on (some of) the quantities. If a configuration is strictly better in quantity A, and A is given priority over quantity B, then that configuration will always be preferred, no matter what quantity B is of the other configuration. Only if the configurations are the same or incomparable on quantity A, quantity B is taken into account. For instance, a shared multimedia infrastructure has to select an optimal configuration in case there are multiple users with different preferences. The users may establish amongst themselves a priority relation indicating the relative importance of their personal preferences (quality as experienced by one user has priority over the quality experience by the other user). This information can be used by the run-time manager to select a concrete working configuration. The priority order does not have to be total, but can also be a partial order ('user A has priority over user B, but between user A and C, we don't know').

Definition 10 *Let $\trianglelefteq$ be a partial ordering on the quantities $\{Q_i \mid 1 \leq i \leq n\}$, giving their relative priorities. Define a cost function $prio : \prod_{i=1}^{n} Q_i \to R$; the quantity R is the same set $\prod_{i=1}^{n} Q_i$, and $prio$ is the identity function, but with the following order: $\bar{r}_1 \preceq_R \bar{r}_2$ iff for every i, $1 \leq i \leq n$, $\bar{r}_1(Q_i) \preceq_{Q_i} \bar{r}_2(Q_i)$ or there is some j, such that $Q_j \trianglelefteq Q_i$ and $\bar{r}_1(Q_j) \prec \bar{r}_2(Q_j)$.*

Note that the structure of the configuration space 'collapses' to a single new quantity (R).

Proposition 2 *The relation $\preceq_R$ as defined in Definition 10 is a partial order and $prio$ is monotone.*

The more priority relations are introduced, the fewer optimal potential solutions remain. When basic quantities are totally ordered in priority, then the configurations will be totally ordered and a unique optimum is obtained. Note also that priorities only add ordering to the configurations; since $prio$ is monotone, if $\bar{c}_1 \preceq \bar{c}_2$, then also $\bar{c}_1 \preceq_R \bar{c}_2$.

Proposition 3 *If $\trianglelefteq$ is a total order and the quantities of the configuration space are basic (totally ordered), then the resulting priority relation $\preceq_R$ is a total order.*

6 Operations of the Algebra

In this section, we introduce operations on sets of configurations. With these operations, we define an algebra of Pareto points that can be used to compute or reason about Pareto points of composite systems. We define the operations, show how they can be applied in practical problems and study their properties, in particular regarding Pareto minimality of operands and results.

6.1 Minimisation

The first operator takes configurations and returns the unique equivalent Pareto minimal set (Pareto frontier).

Definition 11 *Let $\mathcal{C}$ be a set of configurations of configuration space $\mathcal{S}$ and $(\mathcal{C}, \preceq)$ well-ordered, then $\min(\mathcal{C})$ denotes the unique, Pareto equivalent and Pareto minimal set of configurations.*

Its use in practice stems from the fact that a minimal representation of a component's trade-offs makes it easier and more efficient to manipulate, at design-time or at run-time. From Proposition 1 and Theorem 1, the following corollaries follow immediately.

Corollary 2 $\mathcal{C}_1 \equiv \mathcal{C}_2$ *iff* $\min(\mathcal{C}_1) = \min(\mathcal{C}_2)$.

Corollary 3 $\min(\mathcal{C}) \equiv \mathcal{C}$.

Corollary 4 $\min(\mathcal{C}) \subseteq \mathcal{C}$.

Minimisation is the key to keeping the size of a component's description or the number of configurations that we need to consider as small as possible. We want to answer two important questions about the operators we introduce in the remainder. The first asks whether an operator allows minimisation of its operands. Let $\mathrm{f}(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ be an operator with n operands. f supports minimisation of its operands if

$$\mathrm{f}(\mathcal{C}_1, \ldots, \mathcal{C}_n) \equiv \mathrm{f}(\min(\mathcal{C}_1), \ldots, \min(\mathcal{C}_n)) \qquad (1)$$

By minimising operands, which are often intermediate results in the composition process, before applying f, we will not loose optimal configurations. Every practical operator should satisfy property (1), for if this is not the case, then the Pareto minimal set is not an adequate abstraction of the component's configurations. To determine optimal configurations, one would have to try *all* configurations which is often infeasible. If the answer to the first question is positive, the second question is whether an operator preserves minimality. This translates to the following property.

$$\min(\mathrm{f}(\mathcal{C}_1, \ldots, \mathcal{C}_n)) = \mathrm{f}(\min(\mathcal{C}_1), \ldots, \min(\mathcal{C}_n)) \qquad (2)$$

If property (2) holds, then minimisation after applying operator f is unnecessary. The result is always minimal if the operands have been minimised, which saves time in computing the results of the operation. In the remainder, instead of using property (1), we study the somewhat stronger property that the Pareto equivalence is a congruence with respect to the operator. If $\mathcal{C}'_i \equiv \mathcal{C}_i$ for all $1 \leq i \leq n$, then

$$\mathrm{f}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n) \equiv \mathrm{f}(\mathcal{C}_1, \ldots, \mathcal{C}_n). \qquad (3)$$

To show this, it is sufficient to show that the operator preserves dominance. If $\mathcal{C}'_i \preceq \mathcal{C}_i$ for all $1 \leq i \leq n$, then

$$\mathrm{f}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n) \preceq \mathrm{f}(\mathcal{C}_1, \ldots, \mathcal{C}_n). \qquad (4)$$

Property (1) follows directly from (3) or (4), as $\min(\mathcal{C}) \equiv \mathcal{C}$ for any $\mathcal{C}$.

6.2 Free Product

A system often consists of multiple components that each have their own configurations and corresponding trade-offs. To combine these components to systems, we need to derive the configurations and trade-offs of the system from the components. The *free product* describes the combination and assumes that components can be configured entirely independently. Constraints on the composition can be introduced later with other operators.

Definition 12 *Let $\mathcal{C}_1$ be a set of configurations of configuration space $\mathcal{S}_1$ and $\mathcal{C}_2$ a set of configurations of space $\mathcal{S}_2$. Then $\mathcal{C}_1 \cdot \mathcal{C}_2$ is a set of configurations in the configuration space $\mathcal{S}_1 \times \mathcal{S}_2$ that is called the (free) product of $\mathcal{C}_1$ and $\mathcal{C}_2$ and is defined as $\mathcal{C}_1 \cdot \mathcal{C}_2 = \{\bar{c}_1 \cdot \bar{c}_2 \mid \bar{c}_1 \in \mathcal{C}_1, \bar{c}_2 \in \mathcal{C}_2\}$, where $\bar{c} \cdot \bar{d}$ with $\bar{c} = (c_1, \ldots, c_m)$ and $\bar{d} = (d_1, \ldots, d_n)$ is defined as $(c_1, \ldots, c_m, d_1, \ldots, d_n)$.*

The configuration spaces of the server and the network of our example can be combined with the product operator. Assume that the network conditions are currently such that the set of available configurations is $\{nw\_low, nw\_med\}$. The (optimal) server configurations are $\{(A, sq\_low, sb\_low), (B, sq\_med, sb\_med), (C, sq\_high, sb\_high)\}$. The free product then yields:

$$\{(A, sq\_low, sb\_low, nw\_low), (B, sq\_med, sb\_med, nw\_low), (C, sq\_high, sb\_high, nw\_low), (A, sq\_low, sb\_low, nw\_med), (B, sq\_med, sb\_med, nw\_med), (C, sq\_high, sb\_high, nw\_med)\}.$$

Some of these may not be feasible; the network bandwidth must be sufficient to carry the stream. This can be enforced with the constraint operator introduced in Section 6.4.

Proposition 4 *If $\mathcal{C}'_1 \preceq \mathcal{C}_1$, $\mathcal{C}'_2 \preceq \mathcal{C}_2$, then $\mathcal{C}'_1 \cdot \mathcal{C}'_2 \preceq \mathcal{C}_1 \cdot \mathcal{C}_2$.*

As explained in Section 6.1, it follows immediately that the congruence and intermediate minimisation properties hold.

Corollary 5 *If $\mathcal{C}'_1 \equiv \mathcal{C}_1$, $\mathcal{C}'_2 \equiv \mathcal{C}_2$, then $\mathcal{C}'_1 \cdot \mathcal{C}'_2 \equiv \mathcal{C}_1 \cdot \mathcal{C}_2$.*

Corollary 6 $\mathcal{C}_1 \cdot \mathcal{C}_2 \equiv \min(\mathcal{C}_1) \cdot \min(\mathcal{C}_2)$

Additionally, the product operator preserves minimality.

Proposition 5 $\min(\mathcal{C}_1 \cdot \mathcal{C}_2) = \min(\mathcal{C}_1) \cdot \min(\mathcal{C}_2)$

6.3 Alternatives

If configuration sets $\mathcal{C}_1$ and $\mathcal{C}_2$ from configuration space $\mathcal{S}$ describe alternative realisations, then the choice between both realisations is characterised by the set $\mathcal{C}_1 \cup \mathcal{C}_2$. The hardware platform of the hand-held device may have a choice between two processing elements to realise the stream decoder. The different elements may offer different amounts of processing or storage capacity. If we have Pareto sets characterising the performance of the decoders on the individual processing elements, then their union characterises the possible ways in with the hand-held platform can execute the stream decoder.

Proposition 6 *If $\mathcal{C}'_1 \preceq \mathcal{C}_1$, $\mathcal{C}'_2 \preceq \mathcal{C}_2$, then $\mathcal{C}'_1 \cup \mathcal{C}'_2 \preceq \mathcal{C}_1 \cup \mathcal{C}_2$.*

Corollary 7 *If $\mathcal{C}'_1 \equiv \mathcal{C}_1$, $\mathcal{C}'_2 \equiv \mathcal{C}_2$, then $\mathcal{C}'_1 \cup \mathcal{C}'_2 \equiv \mathcal{C}_1 \cup \mathcal{C}_2$.*

Corollary 8 $\mathcal{C}_1 \cup \mathcal{C}_2 \equiv \min(\mathcal{C}_1) \cup \min(\mathcal{C}_2)$.

The alternative does not preserve minimality. It is generally not true that $\min(\mathcal{C}_1 \cup \mathcal{C}_2) = \min(\mathcal{C}_1) \cup \min(\mathcal{C}_2)$. This is the case for instance if mapping the stream decoder to one processing element performs worse (in every aspect) than a mapping to another processing element.

6.4 Constraints

When we have a set of configurations, some of which are invalid because of additional constraints, then we can apply a constraint to filter out only those configurations that satisfy the constraint. Such constraints can be expressed as a set $\mathcal{D}$ of acceptable configurations or equivalently as a proposition on the configuration space, identifying the acceptable configurations. Given a set of configurations, application of the constraint $\mathcal{D}$ amounts to taking the intersection of both sets: $\mathcal{C} \cap \mathcal{D}$ which leaves those configurations of $\mathcal{C}$ satisfying the constraint. There is however one significant problem with this operation. The essential property of congruence is not satisfied. It is not the case in general that $\mathcal{C} \cap \mathcal{D} \equiv \min(\mathcal{C}) \cap \mathcal{D}$. For suppose

that $\mathcal{C} = \{nw\_low, nw\_med, nw\_high\}$ models the available bandwidth of the wireless network and we would like to use $\mathcal{D} = \{nw\_med\}$ as a constraint to capture the fact that the currently available bandwidth is nw_med. In this case, $\mathcal{C} \cap \mathcal{D} = \{nw\_med\}$ and since $\min(\mathcal{C}) = \{nw\_high\}$, $\min(\mathcal{C}) \cap \mathcal{D} = \varnothing$, which are obviously not equivalent. Therefore, minimisation before applying a constraint may result in loss of feasible solutions and such a constraint operator would not be suitable for a framework based on Pareto minimal sets. To remedy this problem, we have to take a closer look at constraints. By minimisation, we have removed configurations that were dominated by another configuration. Therefore, it should never be the case that the dominated configuration passes the constraint while the dominating configuration fails the test. We concentrate on a type of constraints that satisfy this property.

Definition 13 (SAFE CONSTRAINT) *A constraint $\mathcal{D}$ is called a* safe constraint *iff for all $\bar{c}_1, \bar{c}_2$, such that $\bar{c}_1 \preceq \bar{c}_2$, $\bar{c}_2 \in \mathcal{D}$ implies that $\bar{c}_1 \in \mathcal{D}$.*

In other words, a constraint is safe if it is closed under addition of dominating configurations. If a configuration is acceptable, then any configuration that dominates it must be acceptable too. This restriction is quite natural and says that there is no such thing as a configuration that is 'too good'. It is usually satisfied in practice.

When two tasks are mapped onto a single processor, then the sum of the loads λ_1 and λ_2 cannot exceed the capacity p of the processor; i.e., $\lambda_1 + \lambda_2 \leq p$. It is easy to see that such a constraint is safe. If one combination fits and we reduce the load of some task, then the new combination still fits. A constraint in the stream example are that the bandwidth for the stream should fit with the available network bandwidth.

Proposition 7 *Let $\mathcal{C}_1' \preceq \mathcal{C}_1$ and let $\mathcal{D}$ be a safe constraint. Then, $\mathcal{C}_1' \cap \mathcal{D} \preceq \mathcal{C}_1 \cap \mathcal{D}$.*

Corollary 9 *If $\mathcal{C}_1' \equiv \mathcal{C}_1$ and $\mathcal{D}$ is a safe constraint, then $\mathcal{C}_1' \cap \mathcal{D} \equiv \mathcal{C}_1 \cap \mathcal{D}$.*

Corollary 10 *If $\mathcal{D}$ is a safe constraint, then $\mathcal{C} \cap \mathcal{D} \equiv \min(\mathcal{C}) \cap \mathcal{D}$.*

In fact, a safe constraint also preserves minimality.

Proposition 8 *If $\mathcal{D}$ is a safe constraint, then* $\min(\mathcal{C} \cap \mathcal{D}) = \min(\mathcal{C}) \cap \mathcal{D}$.

6.5 Abstraction

Sometimes a configuration space contains too much information; some quantities are not (or no longer) relevant. Abstraction can then be used to remove such a dimension of the configuration space.

Definition 14 *If $\bar{a} = (a_1, \ldots, a_n)$ is a tuple of length n and $1 \leq k \leq n$, then $\bar{a} \downarrow k$ denotes $(a_1, \ldots, a_{k-1}, a_{k+1}, \ldots, a_n)$. Moreover, $\downarrow$ is lifted to sets of tuples as follows: $A \downarrow k = \{\bar{a} \downarrow k \mid \bar{a} \in A\}$.*

Definition 15 (ABSTRACTION) *Let $\mathcal{C}$ be a set of configurations of configuration space $\mathcal{S} = Q_1 \times Q_2 \times \ldots \times Q_n$. Then, $\mathcal{C} \downarrow k$ is a set of configurations over configuration space $\mathcal{S} \downarrow k$. We also write $\mathcal{C} \downarrow Q_k$ to denote $\mathcal{C} \downarrow k$ and $\mathcal{C} \downarrow \{Q_a, Q_b, \ldots\}$ to denote $(\mathcal{C} \downarrow Q_a) \downarrow Q_b \ldots$.*

An example of using the abstraction operator could be the case where the server, the wireless network and the hand-held application have been combined and we have enforced a constraint that the hand-held and the server will use the same mode of video encoding. After this, the actual mode is no longer relevant. It can be abstracted.

Abstraction preserves dominance.

Proposition 9 *If $\mathcal{C}_1' \preceq \mathcal{C}_1$, then $\mathcal{C}_1' \downarrow Q \preceq \mathcal{C}_1 \downarrow Q$.*

Hence, the following corollaries follow.

Corollary 11 *If $\mathcal{C}_1' \equiv \mathcal{C}_1$, then $\mathcal{C}_1' \downarrow Q \equiv \mathcal{C}_1 \downarrow Q$.*

Corollary 12 $\mathcal{C} \downarrow Q \equiv \min(\mathcal{C}) \downarrow Q$

Note however, that a Pareto minimal set may no longer be Pareto minimal after abstraction. Property (2) does not hold. It is not the case, in general, that $\min(\mathcal{C} \downarrow Q) = \min(\mathcal{C}) \downarrow Q$. In the example sketched above, as long as the (unrelated) encoding modes are part of the configuration, a configuration in mode A cannot dominate a configuration in mode B. When we abstract from the mode, we can choose the best configuration amongst all video modes and a configuration in mode A can dominate a configuration in mode B. Therefore, minimisation may be required after abstraction.

7 Compositions of Operators

Using the basic operators of the algebra, we can build more sophisticated compositions.

7.1 Producer-Consumer

Using the previously defined operators, we can compute configurations of a combination of producing and consuming entities. Assume we interpret one dimension of one configuration space as a produced quantity, for instance, the amount of computation power of a processor in the hand-held platform of our example. We can match that to another quantity of another configuration space, which plays the role of the consumer of that quantity. For instance, the video codec application running on the hand-held, requires a certain amount of computational effort in a certain configuration. For the platform, producing more computational power is better. For the codec, it is the other way around. The less computational power it requires, the better it is. The relationship between the two should be constrained such that the platform produces at least the amount of computational effort required by the codec.

This can be realised using a combination of a free product, to combine producer and consumer to a single system, and a constraint to guarantee that the produced and

consumed quantities match. This constraint can be expressed with a mapping function between the producing and the consuming quantities, that is monotonically decreasing. For instance, if the consumed quantity is computational effort expressed for instance in MFlops (millions of floating point operations), and the produced quantity is computational power in 1/MFlops, then this mapping is $p \mapsto \frac{1}{p}$. Note the difference with the computational effort to ensure that smaller is better.

Let $\mathcal{PC} = \mathcal{P} \cdot \mathcal{C}$ be the combined configuration space of producer $\mathcal{P}$ and consumer $\mathcal{C}$ and $f : Q_P \to Q_C$ the mapping between produced quantity Q_P and consumed quantity Q_C. The constraint $\mathcal{M}$ is expressed as follows:

$$\mathcal{M} = \{\bar{c} \in \mathcal{PC} \mid \bar{c}(Q_C) \preceq_{Q_C} f(\bar{c}(Q_P))\}.$$

Note that the constraint $\mathcal{M}$ is indeed safe, because f is monotone decreasing. The Producer-Consumer system is then $\mathcal{PC} \cap \mathcal{M}$. Next, we may want to abstract from the quantities that were matched: Q_P and Q_C.

$$(\mathcal{PC} \cap \mathcal{M}) \downarrow \{Q_P, Q_C\}.$$

7.2 Join

Another useful construct is a join operation, as used in relational databases. The operation can be expressed using the free product and a constraint, the latter stating that certain quantity values be related. A direct computation of the result along the lines of these operators would first construct the entire product space and then discard a lot of configurations where the quantities do not match. It can be computed much more efficiently directly, however. For instance, the server and application in our example need to agree on the quality level of the stream they are communicating.

7.3 The Example Revisited

With the algebra introduced in this paper we continue the example of the wireless video stream system. The hand-held application decodes the video stream. It cannot handle the high quality stream associated with mode C, but can choose between medium and low quality modes B and A. Related to that is the computational effort required for the decoding. $AEffort = \mathbb{N}$, measured, say, in MFlops and obtained by profiling or static analysis. The two configurations of the application are: $\{(A, 10), (B, 50)\}$. mode A for 10 MFlops, mode B for 50. The computation effort is provided by the hand-held platform. The platform can deliver more effort at the expense of a higher energy usage. The two quantities involved are platform effort and power dissipation. $PEffort$ is a number in 1/MFlops. The corresponding power consumption is given in a quantity we call $PPower$, a real number in Watt. The valid platform configurations, showing the trade-off between performance and power consumption, are

$$\{(1/12, 0.23), (1/33, 0.8), (1/75, 1.2)\}.$$

The application runs on the platform and the computation efforts have to be matched (a producer-consumer operation). This combination leads to the following valid configurations:

$$\{(A, 10, 1/12, 0.23), (A, 10, 1/33, 0.8), (A, 10, 1/75, 1.2), (B, 50, 1/75, 1.2)\}.$$

Note that this set is minimal. However, once application and platform have been connected, the computational efforts are not interesting anymore. After abstraction, this gives the configurations showing the video (en)coding mode and corresponding power consumption:

$$\{(A, 0.23), (A, 0.8), (A, 1.2), (B, 1.2)\}$$

in configuration space $EncodingMode \times PPower$. This set of configurations is not minimal. There are three ways to decode mode A, but only the most energy-efficient one is interesting. After minimisation, we get $\{(A, 0.23), (B, 1.2)\}$. Finally, we have to combine the hand-held with the network and server with the constraint that the video modes in the server and the hand-held are matched (a join) and the required bandwidth and available network bandwidth are matched (produced-consumer). It is not difficult to see that this results in the following two optimal configurations:

$$\{(A, sq\_low, sb\_low, nw\_med, A, 0.23), (B, sq\_med, sb\_med, nw\_med, B, 1.2)\}.$$

Abstracting now irrelevant information yields:

$$\{(sq\_low, sb\_low, 0.23), (sq\_med, sb\_med, 1.2)\}.$$

The conclusion of the exercise is that we can select a low quality stream, having low bandwidth usage and low power consumption in the hand-held of 0.23W. The alternative offers medium quality video, which uses more bandwidth of the network and 1.2W of power on the hand-held. If we prioritise quality over power consumption and finally bandwidth, the optimal configuration becomes: $(sq\_med, sb\_med, 1.2)$. Other priorities could lead to the other configuration being preferred. Priorities could also differ over time or depend on system parameters such as remaining battery power. A different configuration may become the better option initiating a system reconfiguration.

8 Computational and Complexity Issues

Pareto sets are formed by combining components and their Pareto sets with the operations of the algebra. The number of Pareto optimal configurations can grow very fast when components are added (in principle at an exponential rate). It is therefore crucial to keep the number of configurations as low as possible, through minimisation (preserving equivalence) or, if necessary, through some form of approximation. One of the advantages of the algebra is that it allows minimisation of the intermediate results of the computation. It also supports abstraction from certain dimensions and the use of cost functions that partially resolve design decisions to reduce the size of the configuration sets.

There are two general, different application scenarios for Pareto analysis and for the algebra. One is the need to compute sets of Pareto points at run-time and potentially

on resource-constrained devices and the other is computing them off-line, at design-time, on a powerful computer for design-space exploration purposes. One of the crucial operations is minimisation of a set of configurations to its Pareto points. The best known algorithm is $\mathcal{O}(N(\log N)^d)$ [2] where d is the number of dimensions of the configuration space and N the number of configurations. The algorithm is quite involved however and is efficient for large N in particular if the number of dimensions is high. The number of points for which the algorithm is faster than simpler algorithms such as the Simple Cull algorithm [16] having a complexity of $\mathcal{O}(N^2)$ is quite high. This makes the algorithm particularly suited for off-line analysis. Hybrid algorithms are also used in this situation [16]. For run-time analysis in resource-constrained devices, the number of configurations has to be limited and the $\mathcal{O}(N^2)$ algorithms are likely to outperform the algorithm of [2]. A similar effect is observed for data structures for storing sets of configurations. The quad-tree data structure [5, 12] is shown to be more efficient than linear lists in [12], but also here, the gain is achieved for large numbers of data points. [16] shows that when keeping points lexicographically sorted, normalised linear tables can be advantageous for computing the union and intersection of sets. Pruning the set of Pareto points may be necessary if it is still too large. The challenge is then to find the best approximation of a set of Pareto points with a limited number of points [10, 18]. We have made a prototype implementation of our algebra, but efficient run-time implementations and complexity issues need further study.

9 Conclusions

In this paper, we have introduced an algebra of Pareto points. The algebra supports and describes the incremental, compositional computation of a set of Pareto-optimal configurations and is based on a rigourous definition of the familiar concepts of Pareto-dominance and Pareto-optimality. We have generalised the Pareto dimensions to partial orders to support compositionality. We have studied the role of cost functions and shown that the generalisation to partial orders makes it possible to use cost functions to make partial design decisions that still leave room for trade-offs to be exploited later. We have given a set of basic operators of the algebra and studied in particular their properties w.r.t. minimality and equivalence to determine whether they can be used effectively in an incremental and compositional design-space exploration. We have shown two examples of useful higher level operations that can be built from the basic operators of the algebra. Future work includes among other things, the study of the use of the algebra to support run-time decision making in dynamic QoS frameworks. Efficient implementations will be indispensable to make this possible on resource-constrained embedded devices.

References

[1] A. Baykasoglu, S. Owen, and N. Gindy. A taboo search based approach to find the Pareto optimal set in multiple objective optimisation. *Journal of Engin. Optimization*, 31:731–748, 1999.

[2] J. Bentley. Multidimensional command-and-conquer. *Communications of the ACM*, 23(4):214–229, April 1980.

[3] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, New York, 2001.

[4] M. Ehrgott and X. Gandibleux. An Annotated Bibliography of Multi-objective Combinatorial Optimization. Technical Report 62/2000, Fachbereich Mathematik, Universität Kaiserslautern, Kaiserslautern, Germany, 2000.

[5] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[6] M. Geilen, T. Basten, B. Theelen, and R. Otten. An algebra of Pareto points. Technical Report ESR-2005-02, Eindhoven University of Technology, January 2005.

[7] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for Pareto-optimal configurations in parameterized system-on-a-chip. *IEEE Trans. VLSI Syst.*, 10(4):416–422, August 2002.

[8] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[9] M. Gries. Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 38(2):131–183, 2004.

[10] C. Mattson, A. Mullur, and A. Messac. Minimal representation of multiobjective design space using a smart Pareto filter. In *Proc. 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002.

[11] V. Pareto. *Manuale di Economia Politica*. Piccola Biblioteca Scientifica, Milan, 1906. Translated into English by A.S. Schwier (1971), Manual of Political Economy, MacMillan, London.

[12] M. Sun and R. E. Steuer. Quad-trees and linear lists for identifying nondominated criterion vectors. *ORSA Journal on Computing*, 8(4):367–375, 1996.

[13] R. Szymanek, F. Catthoor, and K. Kuchcinski. Time-energy design space exploration for multi-layer memory architectures. In *Proc. of DATE 2004*, pages 318–323. IEEE, 2004.

[14] F. Thoen and F. Catthoor. *Modeling, Verification and Exploration of Task-Level Concurrency in Real-Time Embedded Systems*. Kluwer, 2000.

[15] P. Yang and F. Catthoor. Pareto-optimization-based run-time task scheduling for embedded systems. In *Proc. of CODES+ISSS 2003*, pages 120–125, 2003.

[16] M. Yukish. *Algorithms to Identify Pareto Points in Multi-Dimensional Data Sets*. PhD thesis, Pennsylvania State University, August 2004.

[17] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Trans. on Evolutionary Computation*, 3(4):257–271, November 1999.

[18] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. G. D. Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.