Approximating the Pareto Front of Multi-Criteria Optimization Problems[⋆]

Julien Legriel[1,2], Colas Le Guernic[1], Scott Cotton[1], and Oded Maler[1]

[1] CNRS-VERIMAG, 2, av. de Vignate, 38610 Gieres, France
@imag.fr
[2] STMicroelectronics
12, rue Jules Horowitz, 38019 Grenoble, France

Abstract. We propose a general methodology for approximating the Pareto front of multi-criteria optimization problems. Our search-based methodology consists of submitting queries to a constraint solver. Hence, in addition to a set of solutions, we can guarantee bounds on the distance to the actual Pareto front and use this distance to guide the search. Our implementation, which computes and updates the distance efficiently, has been tested on numerous examples.

1 Introduction

Many problems in the design of complex systems are formulated as *optimization* problems, where design choices are encoded as valuations of decision variables and the relative merits of each choice are expressed via a utility/cost function over the decision variables. In most real-life optimization situations, however, the cost function is *multi-dimensional*. For example, a cellular phone that we want to develop or purchase can be evaluated according to its cost, size, power autonomy and performance, and a configuration s which is better than s' according to one criterium, can be worse according to another. Consequently, there is no *unique* optimal solution but rather a set of *efficient* solutions, also known as Pareto[1] solutions, characterized by the fact that their cost cannot be improved in one dimension without being worsened in another. The set of all Pareto solutions, the *Pareto front*, represents the problem trade-offs, and being able to sample this set in a representative manner is a very useful aid in decision making.

Multiple-criteria or multi-objective optimization problems have been studied since the dawn of modern optimization using diverse techniques, depending on the nature of the underlying optimization problems (linear, nonlinear, combinatorial) [10, 4, 5, 3]. One approach consists of defining an aggregate one-dimensional cost/utility function by taking a weighted sum of the various costs. Each choice of a set of coefficients for this sum will lead to an optimal solution for the one-dimensional problem which is also a Pareto solution for the original problem. Another popular class of techniques is

[⋆] This work was partially supported by the French MINALOGIC project ATHOLE

[1] In honor of V. Pareto who introduced them in the context of economic theory [9] to express the fact that different members of society may have different goals and hence social choices cannot be optimal in the one-dimensional sense, a fact consistently ignored in most public debates.

based on heuristic search, most notably genetic/evolutionary algorithms [1, 11], which are used to solve problems related to design-space exploration of embedded systems, the same problems that motivate our work. A major issue in these heuristic techniques is finding meaningful measures of quality for the sets of solutions they provide [12].

In this paper we explore an alternative approach to solve the problem based on *satisfiability/constraint solvers* that can answer whether there is an assignment of values to the decision variables which satisfies a set of constraints. It is well known, in the single-criterium case, that such solvers can be used for optimization by searching the space of feasible costs and asking queries of the form: *is there a solution which satisfies the problem constraints* **and** *its cost is not larger than some constant?* Asking such questions with different constants we obtain both positive (*sat*) and negative (*unsat*) answers. Taking the minimal cost x among the *sat* points and the maximal cost y among the *unsat* points we obtain *both* an approximate solution x and an upper bound $x - y$ on its distance from the optimum, that is, on the quality of the approximation.

In this work we extend the idea to multi-criteria optimization problems. Our goal is to use the *sat* points as an *approximation* of the Pareto front of the problem, use the *unsat* points to guarantee *computable bounds* on the distance between these points and the actual Pareto front and to *direct* the search toward parts of the cost space so as to *reduce* this distance. To this end we define an appropriate metric on the cost space as well as efficient ways to recompute it incrementally as more *sat* and *unsat* points accumulate. A prototype implementation of our algorithm demonstrates the quality and efficiency of our approach on numerous Pareto fronts.

The rest of the paper is organized as follows. Section 2 defines the problem setting including the notions of distance between the *sat* and *unsat* points which guides our search algorithm. In Section 3 we describe some fundamental properties of special points on the boundary of the *unsat* set (*knee points*) which play a special role in computing the distance to the *sat* points, and show how they admit a natural tree structure. In Section 4 we describe our exploration algorithm and the way it updates the distance after each query. Section 5 reports our implementation and experimental results on some purely-synthetic benchmarks of varying dimension and accuracy as well as some scheduling problems where we show the trade-offs between execution time and power consumption. Conclusions and suggestions for future work close the paper.

2 Preliminary Definition

Constrained optimization (we use *minimization* henceforth) problems are often specified as

$$\min c(x) \text{ s.t. } \varphi(x)$$

where x is a vector of decision variables, φ is a set of constraints on the variables that define which solution is considered feasible and c is a cost function defined over the decision variables. We prefer to reformulate the problem by moving costs to the constraint side, that is, letting $\varphi(x, c)$ denote the fact that x is a feasible solution whose cost is c. Hence the optimum is

$$\min\{c : \exists x\ \varphi(x, c)\}.$$

Moving to multi-criteria optimization, c becomes a d-dimensional vector $(c_1, \ldots c_d)$ that we assume, without loss of generality,[2] to range over the bounded hypercube $C = [0,1]^d$, that we call the *cost space*. We use notation $\mathbf{r}$ for $(r, \ldots, r)$.

We assume that the maximal cost $\mathbf{1}$ is feasible and that any cost with some $c_i = 0$ is infeasible. This is expressed as an initial set of *unsat* points $\{\mathbf{0}_i\}_{i=1..d}$ where $\mathbf{0}_i$ is a point with $c_i = 0$ and $c_j = 1$ for every $j \neq i$. The set C is a lattice with a partial-order relation defined as:

$$s \leq s' \equiv \forall i \; s_i \leq s'_i \tag{1}$$

Pairs of points such that $s \not\leq s'$ and $s' \not\leq s$ are said to be *incomparable*, denoted by $s||s'$. The strict version of $\leq$ is

$$s < s' \equiv s \leq s' \wedge \exists j \; s_j < s'_j \tag{2}$$

meaning that s strictly improves upon s' in at least one dimension without being worse on the others. In this case we say that *s dominates s′*. We will make an assumption that if cost s is feasible so is any cost $s' > s$ (one can add a slack variable to the cost). The *meet* and *join* on C are defined as

$$\begin{aligned} s \sqcap s' &= (\min\{s_1, s'_1\}, \ldots, \min\{s_d, s'_d\}) \\ s \sqcup s' &= (\max\{s_1, s'_1\}, \ldots, \max\{s_d, s'_d\}) \end{aligned}$$

We say that a point in the cost space s' is an *i-extension* of a point s if $s'_i > s_i$ and $s'_j = s_j$ for every $i \neq j$.

A point s in a subset $S \subseteq C$ is *minimal* if it is not dominated by any other point in S, and is *maximal* if it does not dominate any point in S. We denote the sets of minimal and maximal elements of S by $\underline{S}$ and $\overline{S}$, respectively. We say that a set S of points is domination-free if all pairs of elements $s, s' \in S$ are incomparable, which is true by definition for $\underline{S}$ and $\overline{S}$. The domination relation associates with a point s two *rectangular cones* $B^+(s)$ and $B^-(s)$ consisting of points dominated by (resp. dominating) s:

$$B^-(s) = \{s' \in C, s' < s\} \;\text{ and }\; B^+(s) = \{s' \in C, s < s'\}.$$

These notions are illustrated in Figure 1. Note that both $B^-(s) \cup \{s\}$ and $B^+(s) \cup \{s\}$ are closed sets. If cost s is feasible it is of no use to look for solutions with costs in $B^+(s)$ because they are not Pareto solutions. Likewise, if s is infeasible, we will not find solutions in $B^-(s)$.[3] We let $B^-(S)$ and $B^+(S)$ denote the union of the respective cones of the elements of S and observe that $B^+(S) = B^+(\underline{S})$ and $B^-(S) = B^-(\overline{S})$.

Suppose that we have performed several queries and the solver has provided us with the sets S_0, and S_1 of *unsat* and *sat* points, respectively. Our state of knowledge is summarized by the two sets $K_1 = B^+(S_1)$ and $K_0 = B^-(S_0)$. We know that K_1 contains no Pareto points and K_0 contains no solutions. The domain for which S_0 and S_1 give us *no information* is $\tilde{K} = (C - K_0) \cap (C - K_1)$. We use $bd(K_0)$ and $bd(K_1)$ to

[2] One can normalize the cost functions accordingly.

[3] Note that the query is formulated as $c \leq s$ and if the problem is discrete and there is no solution whose cost is exactly s, the solver would provide a solution with $c = s' < s$ if such a solution exists.

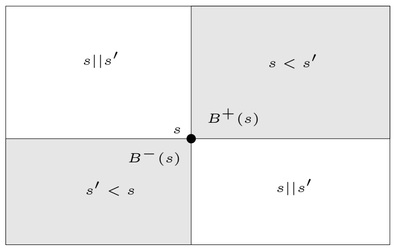

Fig. 1. A point s and its backward and forward cones.

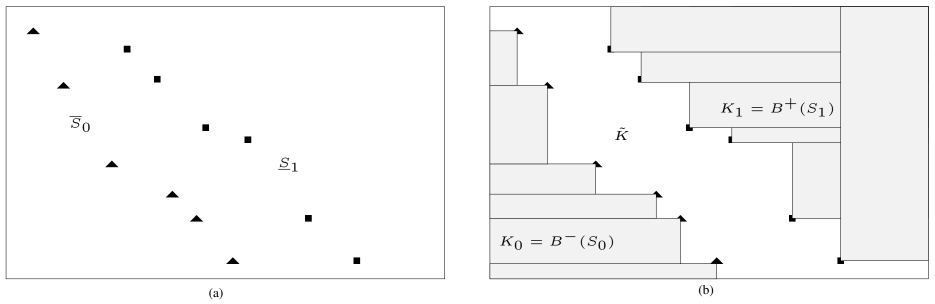

Fig. 2. (a) Sets S_0 and S_1 represented by their extremal points $\overline{S}_0$ and $\underline{S}_1$; (b) The gaps in our knowledge at this point as captured by K_0, K_1 and $\tilde{K}$. The actual Pareto front is contained in the closure of $\tilde{K}$.

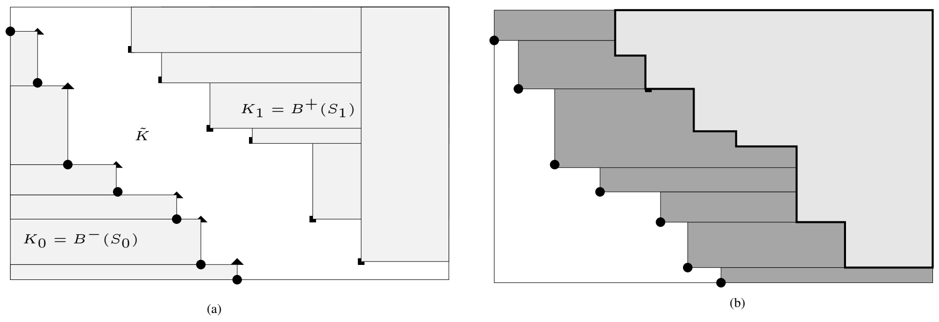

Fig. 3. (a) Knee points, denoted by circles; (b) Knee points viewed as the minimal points of $C - K_0$.

denote the boundaries between $\tilde{K}$ and K_0 and K_1, respectively. It is the "size" of $\tilde{K}$ or the *distance* between the boundaries $bd(K_0)$ and $bd(K_1)$ which determines the quality of our current approximation, see Figure 2. Put another way, if S_1 is our approximation of the Pareto surface, the boundary of K_0 defines the limits of potential improvement of the approximation, because no solutions can be found beyond it. This can be formalized as an appropriate (directed) distance between S_1 and K_0. Note that no point in S_1 can dominate a point in K_0.

Definition 1 (Directed Distance between Points and Sets). *The directed distance $\rho(s, s')$ between two points is defined as*

$$\rho(s, s') = \max\{s'_i \dot{-} s_i : i = 1..d\},$$

where $x \dot{-} y = x - y$ when $x > y$ and 0 otherwise. The distance between a point s and a set S' is the distance between s to the closest point in S':

$$\rho(s, S') = \min\{\rho(s, s') : s' \in S'\}.$$

The Hausdorff directed distance between two sets S and S'

$$\rho(S, S') = \max\{\rho(s, S') : s \in S\}.$$

In all these definitions we assume $s' \not< s$ for any $s \in S$ and $s' \in S'$.

In other words

$$\rho(S, S') = \max_{s \in S} \min_{s' \in S'} \max_{i=1..d} s'_i \dot{-} s_i.$$

Definition 2 (ϵ-Approximation). *A set of points S is an ϵ-approximation[4] of a Pareto front P if $\rho(P, S) \leq \epsilon$.*

Since the Pareto surface is bounded from below by $bd(K_0)$ we have:

Observation 1 *Consider an optimization problem such that S_0 is included in the set of infeasible solutions, with $K_0 = B^-(S_0)$. Then any set S_1 of solutions which satisfies $\rho(bd(K_0), S_1) \leq \epsilon$ is an ϵ-approximation of the Pareto set P.*

Our goal is to obtain an ϵ-approximation of P by submitting as few queries as possible to the solver. To this end we will study the structure of the involved sets and their distances. We are not going to prove new complexity results because the upper and lower bounds on the number of required queries are almost tight:

Observation 2 (Bounds)

1. *One can find an ϵ-approximation of any Pareto front $P \subseteq C$ using $(1/\epsilon)^d$ queries;*
2. *Some Pareto fronts cannot be approximated by less than $(1/\epsilon)^{d-1}$ points.*

[4] This definition is similar to that of [8] except for the fact that their definition requires that for every $p \in P$ there exists $s \in S$ such that $s \leq p + \epsilon p$ and ours requires that $s \leq p + \epsilon$.

Proof. For (1), similarly to [8], define an ϵ-grid over C, ask queries for each grid point and put them in S_0 and S_1 according to the answer. Then take S_1 as the approximation whose distance from $bd(S_0)$ is at most $1/\epsilon$ by construction. For (2), consider a "diagonal" surface

$$P = \{(s_1, \dots, s_d) : \sum_{i=1}^{d} s_i = 1\}$$

which has dimension $d - 1$. ▪

Remark: The lower bound holds for continuous Pareto surfaces. In discrete problems where the solutions are sparse in the cost space one may hope to approximate P with less than $(1/\epsilon)^d$ points, maybe with a measure related to the actual *number* of Pareto solutions. However since we do not work directly with P but rather with S_0, it is not clear whether this fact can be exploited. Of course, even for continuous surfaces the lower bound is rarely obtained: as the orientation of the surface deviates from the diagonal, the number of needed points decreases. A surface which is almost axes-parallel can be approximated by few points.

Updating the distance $\rho(bd(K_0), S_1)$ as more *sat* and *unsat* points accumulate is the major activity of our algorithm hence we pay a special attention to its efficient implementation. It turns out that it is sufficient to compute the distance $\rho(G, S_1)$ where G is a finite set of special points associated with any set of the from $B^-(S)$.

3 Knee Points

Definition 3 (Knee Points). *A point s in $bd(K_0)$ is called a knee point if by subtracting a positive number from any of its coordinates we obtain a point in the interior of K_0. The set of all such points is denoted by G.*

In other words the knee points, illustrated in Figure 3-(a), represent the most *unexplored corners* of the cost space where the maximal potential improvement resides. This is perhaps best viewed if we consider an alternative definition of G as the minimal set such that $C - K_0 = B^+(G)$, see Figure 3-(b). Since $\rho(s, s')$ can only increase as s moves *down* along the boundary we have:

Observation 3 (Distance and Knee Points) $\rho(bd(K_0), S_1)) = \rho(G, S_1)$.

Our algorithm keeps track of the evolution of the knee points as additional *unsat* points accumulate. Before giving formal definitions, let us illustrate their evolution using an example in dimension 2. Figure 4-(a) shows a knee point g generated by two *unsat* points s^1 and s^2. The effect of a new *unsat* point s on g depends, of course, on the relative position of s. Figure 4-(b) shows the case where $s \not> g$: here knee g is not affected at all and the new knees generated are extensions of other knees. Figure 4-(c) shows two *unsat* points dominated by g: point s^5 induces two extensions of g and point s^6 which does not. The general rule is illustrated in Figure 4-(d): s will create an extension of g in direction i iff $s_i < h_i$ where h_i is the extent to which the hyperplane perpendicular to i can be translated forward without eliminating the knee, that is, without taking the intersection of the d hyperplanes outside K_0.

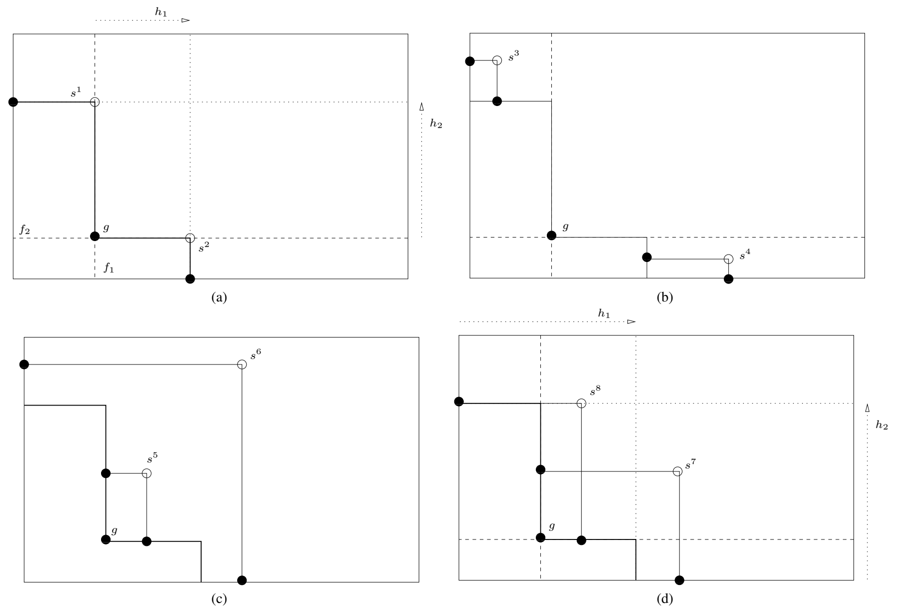

Fig. 4. (a) A knee g generated by s^1 and s^2. It is the intersection of the two hyperplanes f_1 and f_2 (dashed lines); (b) new *unsat* points s^3 and s^4 which are not dominated by g and have no influence on it; (c) new *unsat* points s^5 and s^6 which are dominated by g and hence eliminate it as a knee point. Point s^5 generates new knees as "extensions" of g while the knees generated by s^6 are not related to g; (d) point s^7 generates an extension of g in direction 2 and point s^8 generates an extension in direction 1. These are the directions i where the coordinates of the *unsat* points are strictly smaller than h_i (dotted lines).

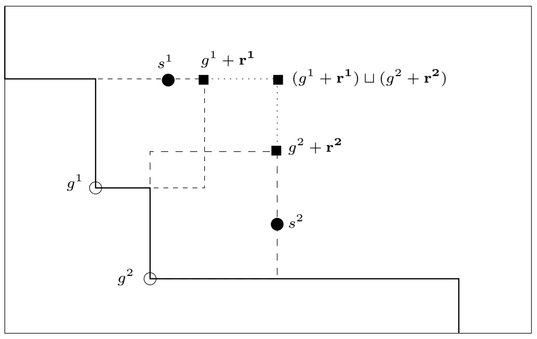

Fig. 5. Two knees g^1 and g^2 and their respective nearest points s^1 and s^2. Points outside the upper dashed square will not improve the distance to g^1 and those outside the lower square will not improve the distance to g^2. Points outside the enclosing dotted rectangle can improve neither of the distances.

Let S be a set of incomparable points and let $\{s^1, \ldots, s^d\} \subseteq S$ be a set of d points such that for every i and every $j \neq i$ $s_i^i \leq s_i^j$. The *ordered meet* of $s^1, \ldots, s^d$ is

$$[s^1, \ldots, s^d] = (s_1^1, s_2^2, \ldots, s_d^d). \tag{3}$$

Note that this definition coincides with the usual meet operation on partially-ordered sets, but our notation is ordered, insisting that s^i attains the minimum in dimension i. The knee points of S are maximal elements of the set of points thus obtained. With every knee $g \in G$ we associate a vector h defined as $h = \langle s^1, s^2, \ldots, s^d \rangle = (h_1, \ldots, h_d)$ with $h_i = \min\limits_{j \neq i} s_i^j$ for every i, characterizing the extendability of s in direction i.

Proposition 1 (Knee Generation). *Let S be a set of* unsat *points with a set of knees G, let s be a new* unsat *point and let G' be the new set of knees associated with $S \cup \{s\}$. Then the following holds for every $g \in G$ such that $g = [s^1, \ldots, s^d]$ and $h = \langle s^1, s^2, \ldots, s^d \rangle$*

1. *Knee g is kept in G' iff $g \not< s$*
2. *If $g \in G - G'$, then for every i such that $s_i < h_i$, G' contains a new knee g', the i-descendant of g, defined as $g' = [s^1, \ldots, s, \ldots, s^d]$, extending g in direction i.*

Before describing the tree data structure we use to represent the knee points let us make another observation concerning the potential contribution of a new *sat* point in improving the minimal distance to a knee or a set of knees.

Observation 4 (Distance Relevance) *Let g, g^1 and g^2 be knee points with $\rho(g, S) = r$, $\rho(g^1, S) = r^1$ and $\rho(g^2, S) = r^2$ and let s be a new* sat *point. Then*

1. *The distance $\rho(g, s) \leq r$ iff $s \in B^-(g + \mathbf{r})$;*
2. *Point s cannot improve the distance to any of $\{g^1, g^2\}$ if it is outside the cone $B^-((g^1 + \mathbf{r^1}) \sqcup (g^2 + \mathbf{r^2}))$.*

Note that for the second condition, being in that cone is necessary but not sufficient. The sufficient condition for improving the distance of at least one of the knees is $s \in B^-((g^1 + \mathbf{r^1}) \cup (g^2 + \mathbf{r^2}))$ as illustrated in Figure 5.

We represent G as a tree whose nodes are either leaf nodes that stand for current knee points, or other nodes which represent points which were knees in the past and currently have descendant knees that extend them. A node is a tuple

$$N = (g, [s^1, \ldots, s^k], h, (\mu^1, \ldots, \mu^k), r, b)$$

where g is the point, $[s^1, \ldots, s^k]$ are its *unsat* generators and h is the vector of its extension bounds. For each dimension i, μ^i points to the i-descendant of N (if such exists) and the set of all direct descendants of N is denoted by μ. For leaf nodes $N.r = \rho(N.g, S_1)$ is just the distance from the knee to S_1 while for a non-leaf node $N.r = \max_{N' \in N.\mu} N'.r$, the maximal distance to S_1 over all its descendants. Likewise $N.b$ for a leaf node is the maximal point such that any *sat* point in the interior of its back cone improves the distance to $N.g$. For a non leaf node $N.b = \bigsqcup_{N' \in N.\mu} N'.b$, the join of the bounds associated with its descendants.

4 The Algorithm

The following iterative algorithm submits queries to the solver in order to decrease the distance between S_1 and G.

Algorithm 1 (Approximate Pareto Surface)

```
initialize
repeat
  select(s)
  query(s)     %     ask whether there is a solution with cost $\leq s$
  if sat
    update-sat(s)
  else
    update-unsat(s)
until $\rho(G, S_1) < \epsilon$
```

The initialization procedure lets $S_0 = \{\mathbf{0}_1, \ldots, \mathbf{0}_d\}$, $S_1 = \{(1, \ldots, 1)\}$ and hence initially $G = \{g^0\}$ with $g^0 = [\mathbf{0}_1, \ldots, \mathbf{0}_d] = (0, \ldots, 0)$ and $h = \langle \mathbf{0}_1, \ldots, \mathbf{0}_d \rangle = (1, \ldots, 1)$. The initial distance is $\rho(G, S_1) = 1$. The *update-sat* and *update-unsat* procedures recompute distances according to the newly observed point by propagating s through the knee tree. In the case of a *sat* point, the goal is to track the knee points g such that $\rho(g, s) < \rho(g, S_1)$, namely points whose distance has decreased due to s. When s is an *unsat* point, we have to update G (removing dominated knees, adding new ones), compute the distance from the new knees to S_1 as well as the new maximal distance. The algorithm stops when the distance is reduced beyond ϵ. Note that since $\rho(G, S_1)$ is maintained throughout the algorithm, even an impatient user who aborts the program before termination will have an approximation guarantee for the obtained solution.

The propagation of a new *sat* point s is done via a call to the recursive procedure *prop-sat*(N_0, s) where N_0 is the root of the tree.

Algorithm 2 (Prop-Sat)

```
proc prop-sat(N, s)

if $s < N.b$     %     s may reduce the distance to N.g or its descendants
  $r := 0$     %     temporary distance over all descendants
  $b := \mathbf{0}$     %  temporary bound on relevant sat points
  if $N.\mu \neq \emptyset$     %  a non-leaf node
    for every i s.t. $N' = N.\mu^i \neq \emptyset$ do     %  for every descendant
      prop-sat$(N', s)$
      $r := \max\{r, N'.r\}$
      $b := b \sqcap N'.b$
  else     %  leaf node
    $r := \min\{N.r, \rho(N.g, s)\}$     %  improve if s is closer
    $b := N.g + \mathbf{r}$
  $N.r := r$
  $N.b := b$
```

The propagation of a new *unsat* point s, which is more involved, is done by invoking the recursive procedure *prop-unsat*(N_0, s). The procedure returns a bit ex indicating whether the node still exists after the update (is a knee or has descendants).

Algorithm 3 (Prop-Unsat)

```
proc prop-unsat(N, s)

$ex := 1$
if $N.g < s$      % knee is influenced
   $ex := 0$      % temporary existence bit
   $r := 0$      % temporary distance over all descendants
   $b := \mathbf{0}$      % temporary relevance bound
   if $N.\mu \neq \emptyset$      % a non-leaf node
      for every $i$ s.t. $N' = N.\mu^i \neq \emptyset$ do      % for every descendant
         $ex' :=$ prop-unsat$(N', s)$
         if $ex' = 0$
            $N.\mu^i := \emptyset$      % node $N'$ is removed
         else
            $ex := 1$
            $r := \max\{r, N'.r\}$
            $b := b \sqcup N'.b$
   else      % leaf node
      for $i = 1..d$ do
         if $s_i < N.h_i$      % knee can extend in direction $i$
            $ex := 1$
            create a new node $N' = N.\mu^i$ with
               $N'.g = [N.s^1, \ldots, s, \ldots, N.s^k]$
               $N'.h = \langle N.s^1, \ldots, s, \ldots, N.s^k \rangle$
               $N'.r = \rho(N'.g, S_1)$
               $N'.b = N'.g + N'.r$
               $N'.\mu^i = \emptyset$ for every $i$
            $r := \max\{r, N'.r\}$
            $b := b \sqcup N'.b$
   $N.r := r$
   $N.b := b$
return$(ex)$
```

The *prop-unsat* procedure has to routinely solve the following sub problem: given a knee point g and a set of non-dominating *sat* points S, find a point $s \in S$ nearest to g and hence compute $\rho(g, S)$. The distance has to be non negative so there is at least one dimension i such that $g_i \leq s_i$. Hence a lower bound on the distance is

$$\rho(g, S) \geq \min\{s_i - g_i : (i = 1..d) \wedge (s \in S) \wedge (s_i \geq g_i)\},$$

and an upper bound is:

$$\rho(g, S) \leq \max\{s_i - g_i : (i = 1..d) \wedge (s \in S)\}.$$

We now present an algorithm and a supporting data structure for computing this distance. Let $(L_i, <_i)$ be the linearly-ordered set obtained by projecting $S \cup \{g\}$ on dimension i. For every $v \in L_i$ let $\Theta_i(v)$ denote all the points in S whose i^{th} coordinate is v. Let $\sigma^i(s)$ be the successor of s_i according to $<_i$, that is, the smallest s'_i such that $s_i < s'_i$. Our goal is to find the minimal value v in some L_i such that for every $s \in \Theta_i(v)$, s_i defines the maximal distance to g, that is, $s_i - g_i > s_j - g_j$ for every $j \neq i$.

The algorithm keeps a frontier $F = \{f_1, \ldots, f_d\}$ of candidates for this role. Initially, for every i, $f_i = \sigma^i(g_i)$, the value next to g_i and the candidate distances are kept in $\Delta = \{\delta_1, \ldots, \delta_d\}$ with $\delta_i = f_i - g_i$. The algorithm is simple: each time we pick the minimal $\delta_i \in \Delta$. If for some $s \in \Theta_i(f_i)$ and for every $j \neq i$ we have $s_j - g_j < s_i - g_i$ then we are done and found a nearest point s with distance δ_i. Otherwise, if every $s \in \Theta(f_i)$ admits some j such that $s_j - g_j > s_i - g_i$ we conclude that the distance should be greater than δ_i. We then let $f_i = \sigma^i(f_i)$, update δ_i accordingly, take the next minimal element of Δ and so on. This procedure is illustrated in Figure 6. The projected order relations are realized using an auxiliary structure consisting of d linked lists.

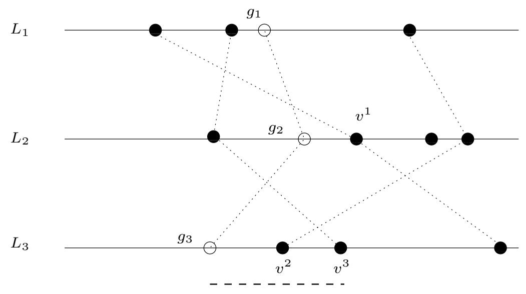

Fig. 6. Finding the nearest neighbor of g: the first candidate for the minimal distance is v^1, the nearest projection which is on dimension 2 , but the point associated with it has a larger distance on dimension 3; The next candidate is v^2, the closest in dimension 3 but the corresponding point also has larger coordinates. Finally, the point associated with v^3, the next value on L_3, has all its distances in other dimensions smaller and hence it is the closest point which defines the distance (dashed line).

Selecting the next query to ask is an important ingredient in any heuristic search algorithm, including ours. We currently employ the following simple rule. Let g and s be a knee and a *sat* point whose distance $\rho(g, s)$ is maximal and equal to $r = s_i - g_i$ for some i. The next point for which we ask a query is $s' = s + \mathbf{r}/2$. If s' turns out to be a *sat* point, then the distance from g to S_1 is reduced by half. If s' is an *unsat* point then g is eliminated and is replaced by zero or more new knees, each of which is r-closer to S_1 in one dimension. For the moment we do not know to compute an upper bound on the worst-case number of queries needed to reach distance ϵ except for some hand-waving arguments based on a discretized version of the algorithm where queries are restricted to the ϵ-grid. Empirically, as reported below, the number of queries was significantly smaller than the upper bound.

5 Experimentation

We have implemented Algorithm 1 and tested it on numerous Pareto fronts produced as follows. We generated artificial Pareto surfaces by properly intersecting several convex and concave halfspaces generated randomly. Then we sampled $10,000$ points in this surface, defined the Pareto front as the boundary of the forward cone of these points and run our algorithm for different values of ϵ. Figure 7 shows how the approximate solutions and the set of queries vary with ϵ on a 2-dimensional example. One can see that indeed, our algorithm concentrates its efforts on the neighborhood of the front. Table 1 shows some preliminary results obtained as follows. For every dimension d we generate several fronts, run the algorithm with several values of ϵ, compute the average number of queries and compare it with the upper bound $(1/\epsilon)^d$. As one can see the number of queries is only a small fraction of the upper bound. Note that in this class of experiments we do not use a constraint solver, only an oracle for the feasibility of points in the cost space based on the generated surface.

d	no tests	ϵ	$(1/\epsilon)^d$	min no queries	avg no queries	max no queries
2	40	0.050	400	5	11	27
		0.025	1600	6	36	111
		0.001	1000000	21	788	2494
3	40	0.050	8000	5	124	607
		0.025	64000	6	813	3811
	20	0.002	125000000	9	30554	208078
4	40	0.050	160000	5	1091	5970
		0.025	2560000	10	11560	46906

Table 1. The average number of queries for surfaces of various dimensions and values of ϵ.

We also have some preliminary results on the following problem which triggered this research: given an application expressed as a task-data graph (a partially-ordered set of tasks with task duration and inter-task communication volume) and a heterogenous multi-processor architecture (a set of processors with varying speeds and energy consumptions, and a communication topology), find a mapping of tasks to processors and a schedule so as to optimize some performance criteria. In [6] we have used the SMT solver Yices [2] to solve the single-criterium problem of finding the cheapest (in terms of energy) configuration on which such a task graph can be scheduled while meeting a *given* deadline. We applied our algorithm to solve a multi-criteria version of the problem, namely to show trade offs between energy cost and execution time. We were able to find 0.05-approximations of the Pareto front for problem with up to 30 tasks on an architecture with 8 processors of 3 different speeds and costs. The behavior of our algorithm is illustrated in Figure 8 where both execution time and energy are normalized

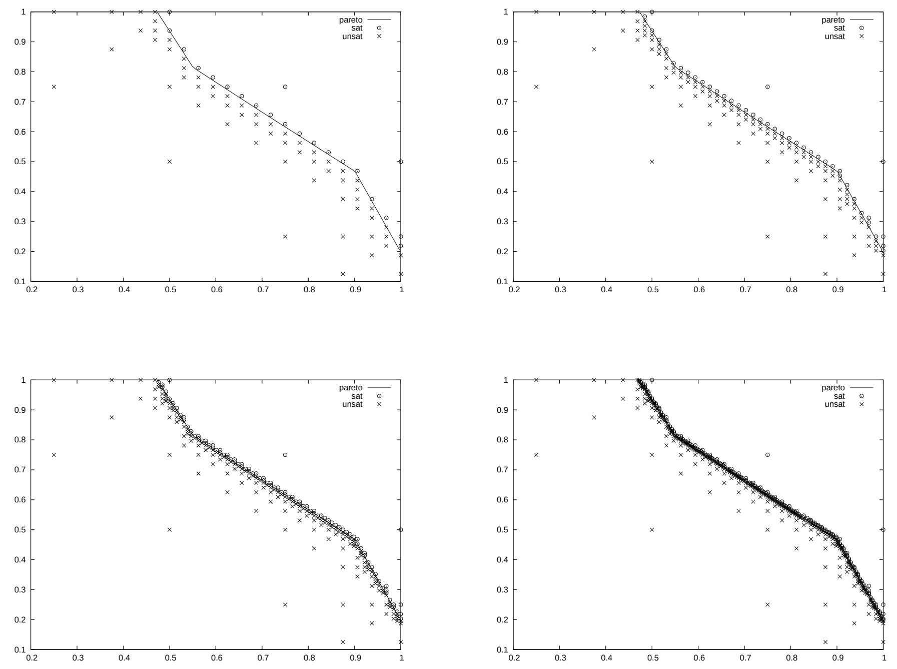

Fig. 7. The results of our algorithm for the same front for $\epsilon = 0.05, 0.125, 0.001, 0.0005$.

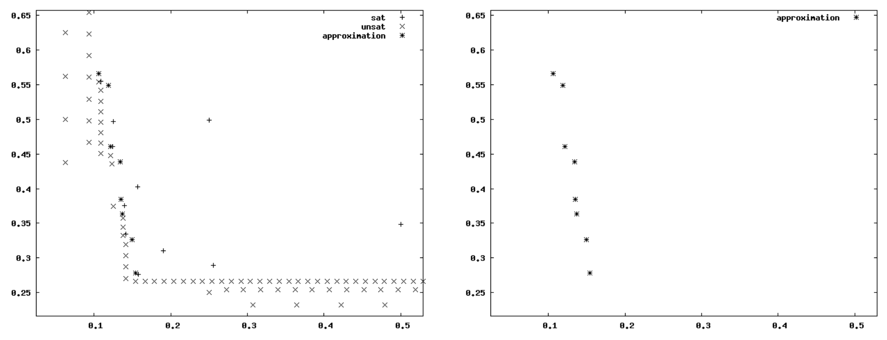

Fig. 8. Scheduling a task-data graph of 20 tasks on an architecture with 8 processors with 3 different levels of speed/consumption: (a) the queries asked; (b) the final front approximation (makespan is horizontal and energy cost is vertical).

to $[0, 1]$. For reasons discussed in the sequel, it is premature to report the computational cost of the algorithm on these examples.

6 Conclusions

We have presented a novel approach for approximating the Pareto front. The difficulty of the problem decomposes into two parts which can, at least to some extent, be decoupled. The first is related to the hardness of the underlying constraint satisfaction problem, which can be as easy as linear programming or as hard as combinatorial or nonlinear optimization. The second part is less domain specific: approximate the boundary between two mutually-exclusive subsets of the cost space which are not known a priori, based on adaptive sampling of these sets, using the constraint solver as an oracle. We have proposed an algorithm, based on a careful study of the geometry of the cost space, which unlike some other approaches, provides *objective* guarantees for the quality of the solutions in terms of a bound on the approximation error. Our algorithm has been shown to behave well on numerous examples.

The knee tree data structure represents effectively the state of the algorithm and reduces significantly the number of distance calculations per query. We speculate that this structure and further geometrical insights can be useful as well to other approaches for solving this problem. We have investigated additional efficiency enhancing tricks, most notably, *lazy* updates of the knee tree: if it can be deduced that a knee g does not maximize the distance $\rho(S_0, S_1)$, then its distance to it $\rho(g, S_1)$ need not be updated in every step. Many other such improvement are on our agenda.

In the future we intend to investigate specializations and adaptations of our general methodology to different classes of problems. For example, in convex linear problems the Pareto front resides on the surface of the feasible set and its approximation may benefit from convexity and admit some symbolic representation via inequalities. More urgently, for hard combinatorial and mixed problems, such as mapping and scheduling, where computation time grows drastically as one approaches the Pareto front, we have to cope with computations that practically do not terminate. We are developing a variant of our algorithm with a limited time budget per query where in addition to *sat* and *unsat*, the solver may respond with a *time-out*. Such an algorithm will produce an ϵ-approximation of the best approximation obtainable with that time budget per query. Adding this feature will increase the size of mapping and scheduling problems that can be robustly handled by our algorithm. To conclude, we believe that the enormous progress made during the last decade in SAT and SMT solvers will have a strong impact on the optimization domain [7] and we hope that this work can be seen as an important step in this direction.

References

1. K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.
2. B. Dutertre and L.M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94, 2006.
3. M. Ehrgott. *Multicriteria optimization*. Springer Verlag, 2005.

4. M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000.
5. J. Figueira, S. Greco, and M. Ehrgott. *Multiple criteria decision analysis: state of the art surveys*. Springer Verlag, 2005.
6. J. Legriel and O. Maler. Meeting deadlines cheaply. Technical Report 2010-1, VERIMAG, January 2010.
7. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006.
8. C.H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92, 2000.
9. V. Pareto. Manuel d'économie politique. *Bull. Amer. Math. Soc.*, 18:462–474, 1912.
10. R.E. Steuer. *Multiple criteria optimization: Theory, computation, and application*. John Wiley & Sons, 1986.
11. E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.
12. E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.