Bi-objective Pareto frontier approximation: A graphical result to an embarrassingly parallel problem

Peter Cacioppi (pcaciopp@us.ibm.com), Scientist, Software Engineer, IBM Software Group, Commerce, IBM

May 2013

Summary: The bi-objective Pareto frontier is easily understood as a trade-off curve for bi-objective mixed integer programming (MIP) optimization problems. Computing such a frontier is computationally intensive. Often, compromises are made and ad-hoc techniques, such as blended objectives or what-if analysis, are used instead. This article surveys bi-objective Pareto frontier approximation and explains how this problem can be easily parallelized into single objective sub-problems which can take advantage of the availability of multi-core machines and highly parallelized cloud compute resources.

With the V7.2 release in the fall of 2011, IBM® LogicNet Plus® (LNP) became the first off-the-shelf optimization tool to incorporate true bi-objective optimization. By employing a variety of graphical and mathematical techniques, LNP was effectively able to extend the optimization power of IBM CPLEX® Optimizer (CPLEX) into two objective dimensions. This enabled a broad spectrum of users to effectively analyze the trade-offs between two competing goals.

This article will survey the following topics

1. We will discuss why bi-objective optimization is a compelling business opportunity. This problem combines easily parallelized computational techniques with a graphical and intuitive result. It is well suited to expand the optimization market while exploiting the computational potential of cloud computing. Other MIP feature advancements run the risk of being obscured by a lack of customer expertise. By contrast, a bi-objective Pareto frontier is easily recognized as a trade-off curve, which is one of the most common visual results of data analysis.

2. We discuss bi-objective optimization as it is seen by LNP users. We demonstrate how a clean "black-box" separation exists between the intuitive graphical result and the underlying formal mathematics. A variety of PhD level computational techniques are surveyed, including a new, highly parallelizable algorithm for enumerating a discrete Pareto frontier. All of these techniques create a visual and intuitive result set.

3. We will formally describe the bi-objective optimization problem and the *L,U* approximation sets that are generated as a result. We will explain how both the *U* set and the *L* set can be created in a parallelized and robust fashion. Sub-problems results can be processed in different orderings, and the loss of a single sub-problem does not destroy the entire result set.

4. We summarize a new Pareto frontier enumeration algorithm that advances the current state of the art. This algorithm is well suited to cloud computing in that it focuses on simple sub-problems that can be solved in parallel. It can be efficiently hot-started by brute force techniques. If terminated prior to completion, it can return a high quality approximation of the Pareto frontier.

Why Pareto? Why now?

Before we discuss the LNP bi-objective feature in detail, it might be worth asking "why bother computing this result at all?" That is a reasonable question. In short, Pareto frontier analysis has the potential to combine cutting edge computing power with an intuitive result set. We briefly describe both of these attractions.

Enabled by modern technology

Optimization technology has historically been used to solve single-objective problems. For example - "find the cheapest long term supply chain strategy without spending more than $10 million in upfront costs". These mixed integer programming problems have historically been so difficult that users were happy enough to simply find a near-optimal solution.

However, as both optimization algorithms and desktop hardware have improved, these single objective problems have required less and less computation time to solve. Thus, the idea of employing an algorithm that relies on multiple single-objective sub-problems has become more realistic.

Even more profound, however, is the potential of cloud computing. Single-objective problems are notoriously stubborn to massive parallelization. In general, throwing 10 parallel processors at a single objective problem yields far less than a 10 fold improvement in running time. Even worse, the algorithmic details of single-objective optimization do not lend themselves to the massive parallelization of truly distributed computing. The underlying combinatorics of mixed integer programming can easily overwhelm any brute-force approach. It is unlikely that we will see, for example, 100 distinct cloud machines working in concert to consistently achieve a 20 fold performance improvement for single objective optimization problems.

Bi-objective optimization, however, is completely different. Solving 2 initialization problems will bound the result set to within a reasonable span. It would then be quite easy to break this span into 100 sub-intervals, and solve 100 independent, single objective sub-problems in parallel. The resulting data set, while not necessarily enumerating the Pareto frontier, would create a highly accurate approximation.

This approximation could then be fed back into the cloud to enumerate the exact Pareto frontier in parallel, if such precision were required. Of course, not every single-objective MIP can be solved to full optimality, and thus not every bi-objective Pareto frontier can be precisely mapped. However, for those bi-objective problems which yield solvable single-objective sub-problems, it is likely that a perfect mapping of the bi-objective Pareto frontier can be generated by a massively parallel compute engine.

In summary, both approximating and mapping the Pareto frontier are highly seperable problems. Although the label “embarrassingly parallel” (see Resources) is perhaps a modest exaggeration, it is hard to imagine a combinatorial optimization problem that is better poised to exploit the emergence of cheap, accessible and massively parallel computing in the global marketplace.

Intuitive results

The nuances of various optimization improvements are often lost on users. The number of people who understand the rules of convex optimization, and how it relates to mixed integer quadratic programming, is fairly small. The number of people who can appreciate when the functionality of the latter is extended to more closely approximate the former is even smaller.

On the other hand, almost everyone understands a trade-off curve. Pareto frontier approximation is simply providing a graphical representation of the shop-worn aphorism, “there is no free lunch.”

Moreover, it is almost certain that a significant portion of single objective optimization models employ some form of implicit multi-objective analysis. Tactical users might be grafting two objective functions into one by adding them together. This surprisingly common practice makes the fundamental mistake of “mixing apples and oranges.” Strategic users often perform crude “what-if” analyses to capture the general shape of the Pareto frontier. Users straddling these two extremes likely use some combination of these techniques. All of these situations could benefit from an automated Pareto frontier approximation tool, particularly one that can be turbo-charged by the cloud.

Usage of the Pareto frontier approximation

It is important to note that Pareto frontier approximations have a very clear “black box barrier.” That is to say, much like single objective mixed integer programming, this problem exhibits a nice separation between the interesting mathematics required for implementation and the relatively straightforward concepts required for application. With a decent graphical user interface, a professional modeler need not be aware of terms like *L,U* approximation sets, Chebyshev distance, or tolerance vector. Rather, a user interacts with much more straightforward concepts, such as competing goals, achievable frontier, and impossible frontier.

- **Competing goals** - In general, the designation of a single objective function is a mathematical convenience introduced by computational programmers. Professional modelers are fully aware of the multiple, often competing, metrics with which to measure the fitness of a proposed solution. These users do not need to be convinced of the utility of a “trade-off” curve. Our training focuses on the likely intractability of

computing a precise trade-off curve. This problem is ameliorated by bounding the curve from above and below. Hence, competing goals are naturally visualized as two distinct frontiers, which approach each other as the algorithm progresses.

- **Achievable frontier** - This line is quite comprehensible to a wide range of users. It represents the best known solutions. For a bi-objective minimization problem, this curve will move downward and to the left as the algorithm discovers solutions that are cheaper in at least one dimension.

- **Impossible frontier** - This line is perhaps a bit more nuanced, as it represents a limit to optimization. Like the "best possible" update CPLEX provides for single objective models, it improves by becoming more expensive. In this case, these improvements are visualized by the curve moving upwards and to the right in a dual-minimization problem.

Thus, the algorithm has an intuitive graphical appeal. It begins with a crude approximation to the Pareto frontier that is drawn as a broad rectangle. The achievable frontier appears in green as the upper and right hand portions, and the impossible frontier is in red as the lower and left hand portions. As the algorithm progresses, the single broad rectangle is replaced with a more detailed "tiling" of smaller rectangles. Eventually, the two frontiers move quite close to each other, and the location of the true Pareto points becomes apparent.

We illustrate this below with an example taken from Chapter 11 of *Supply Chain Network Design* (see Resources). In this case, a supply chain planner is considering the trade-off between initial and long term expenditures (upfront costs versus total costs). This is a likely trade-off for a wide range of optimization problems. On the one hand, long term savings generally require significant initial investments. On the other hand, such investments, once made, represent sunk costs that will be of little use in the event of unexpected and dramatic changes to the target environment. It is natural for the user to seek the cheapest solution, while at the same time limiting his exposure to the risks inherent in large upfront costs.

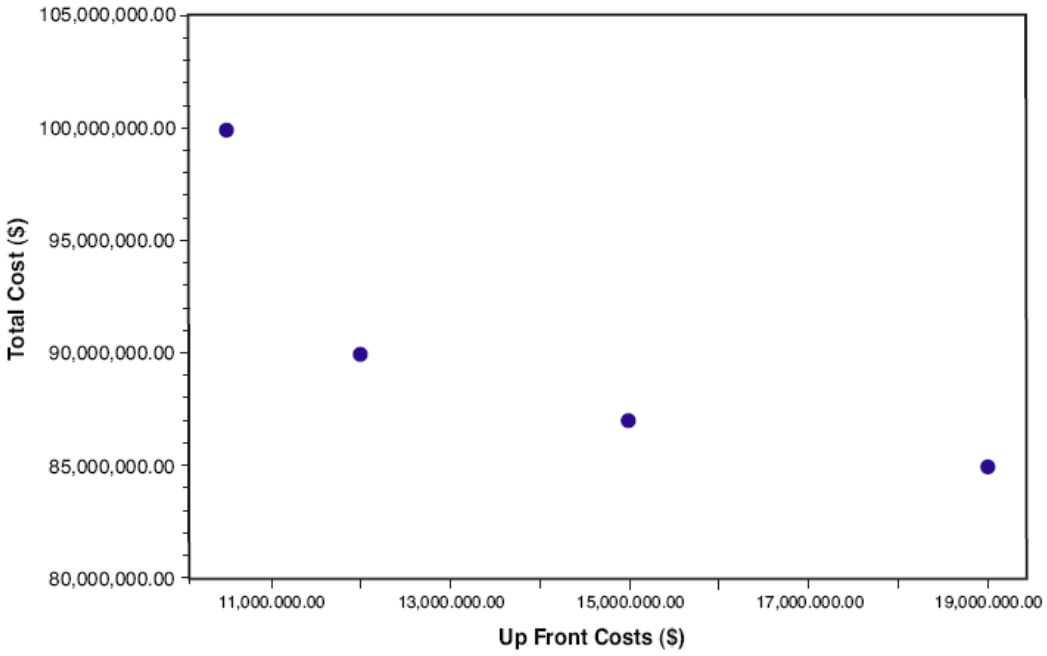
In this example, we first motivate the reader by introducing the concept of Pareto optimal solutions. These solutions are easily explained as being those which allow for alternative compromises, but not for simple improvements. That is to say, if a supply chain plan is Pareto optimal, then it is impossible to decrease its upfront cost without increasing its total cost, and vice versa. It is axiomatic that Pareto optimal solutions (or solutions that are very close to Pareto optimal) will be the only ones worthy of consideration if the two metrics of upfront cost and total cost are paramount.

The Pareto frontier can then be explained as simply a plotting of all possible Pareto optimal solutions. For some models, there are an infinite number of Pareto optimal solutions (in the same way that there are an infinite number of recipes for mixing gin and vermouth) and the

Pareto frontier roughly resembles a curve. For other models, there are only a finite number of Pareto optimal solutions (in the same way there are only a finite number of car models to choose from) and the Pareto frontier resembles a scattering of points.

However, even in this case it is worthwhile to draw two curves – one representing what is possible, and the other representing what isn't. For example, suppose our supply chain model has many possible solutions, but only 4 Pareto solutions. We plot this Pareto frontier in Figure 1 below:

Figure 1 – A tiny example of a Pareto frontier



A common challenge for a user is to apply a strategy for discovering these four points without an automated tool capable of plotting (or approximating) the Pareto frontier. The typical approach involves using what-if analysis with single-objective optimization tools. In this case, a user might choose to optimize the total cost for three scenarios with three different upfront cost restrictions: "upfront cost <= $10 million," "upfront cost <= $15 million," and "upfront cost <= $20 million." These results could easily play out as follows.

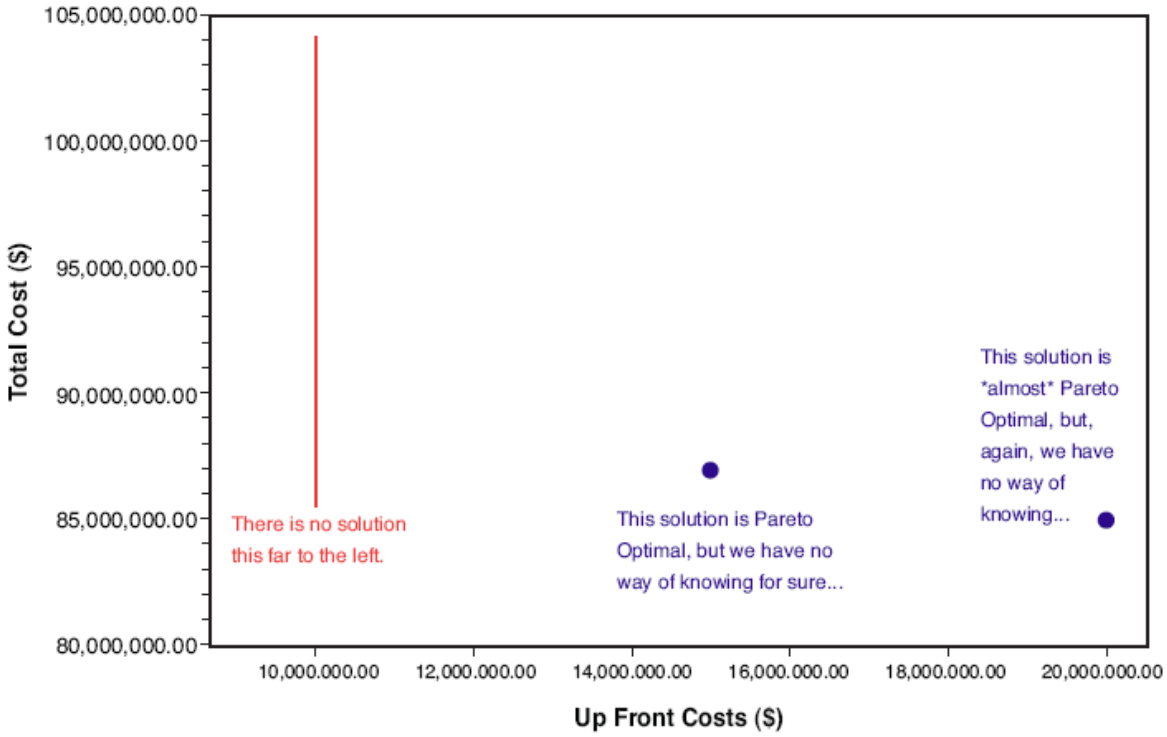
Scenario 1 is infeasible, because the upfront restriction was too tight. That is, there is no solution whose upfront cost is less than or equal to $10 million. Ideally, this infeasibility would be well diagnosed by a sensible error message, but in general, infeasibility diagnosis is quite difficult.

Scenario 2 might generate our third Pareto solution with “upfront = $15 million” and “total = $87 million”. However, the user would have no evidence that this solution is truly Pareto optimal.

Scenario 3 might generate a solution that is “almost Pareto”; that is, “upfront cost = $20 million” and “total cost = $85 million”.

We graph this outcome below in Figure 2:

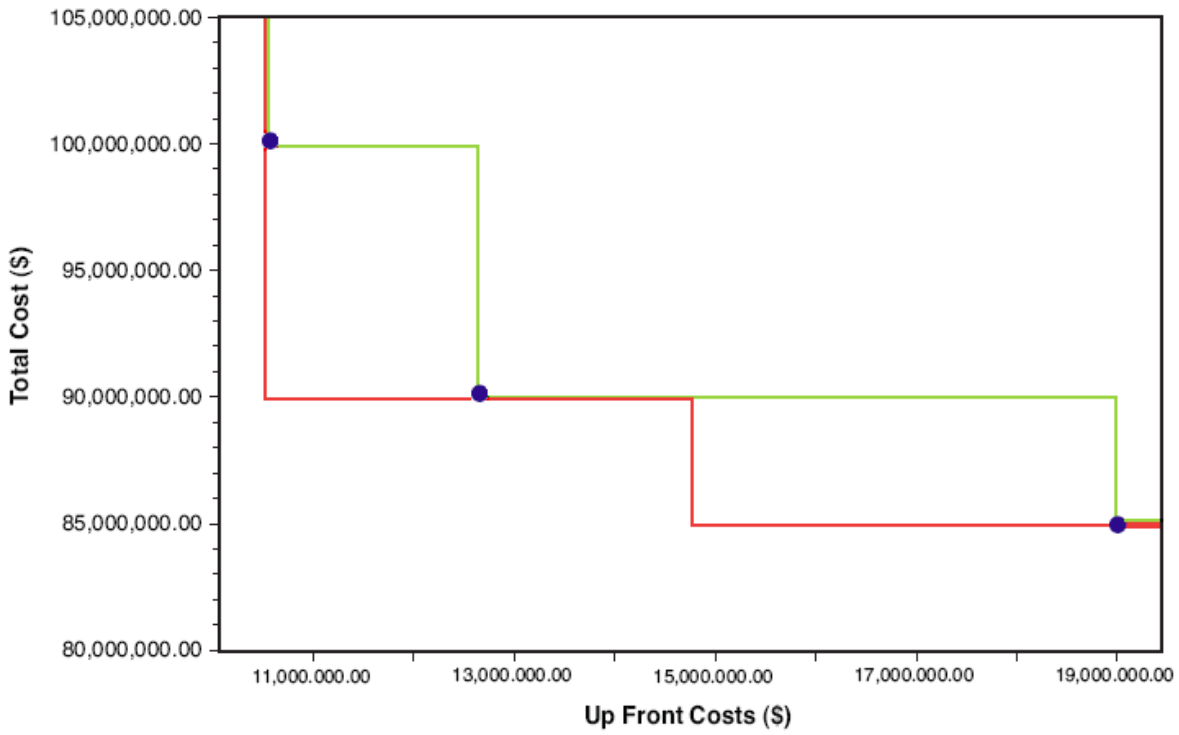
Figure 2 : What-if analysis results



Essentially, these two graphs are enough to motivate the development of an automated Pareto frontier approximation tool. It is not at all uncommon for a user to wish to explore the points on the Pareto frontier. However, the current single-objective optimization tools provide no reliable method for doing so. Although users need some method of discovering Figure 1, they are currently capable only of creating variations on Figure 2.

The bi-objective functionality in LNP addresses this shortcoming nicely. The following graph, resulting from the internal optimization of three sub-models, provides an initial estimate of where the true Pareto frontier is located.

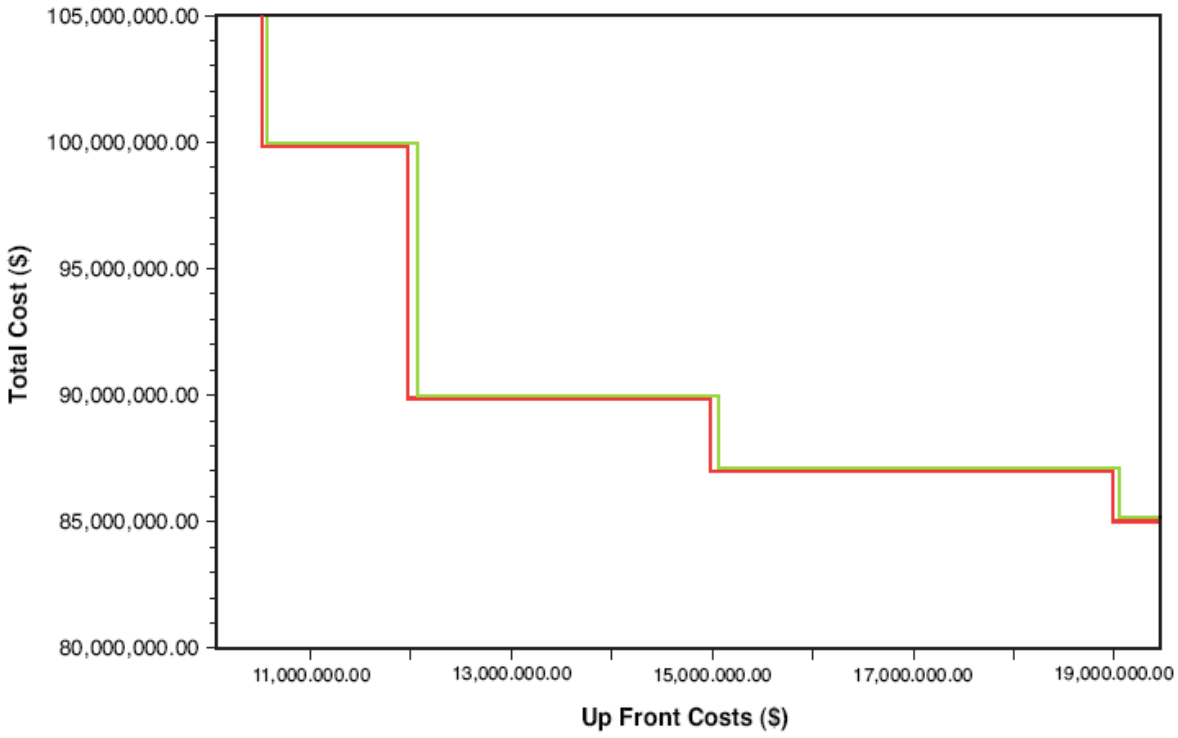
Figure 3 : A crude Pareto frontier approximation



The red line represents the limits to optimization. Any feasible solution, when plotted by its upfront and total costs, will lie above and to the right of the red line. Any solution whose plot (or bi-objective projection) lies above and to the right of the green line must not be Pareto, because it can be improved upon by replacing it with one of the dark-blue dots which corner the green line. That is to say, the green line identifies the portion of the graph which contains points clearly inferior to one of the three blue dots, which in turn plot the best solutions we have found so far.

With an additional four sub-models, the shape of the Pareto curve becomes quite apparent. Although this graph is technically only an approximation to the Pareto frontier, it represents, for all practical purposes, a perfect enumeration of the tradeoffs between total and upfront costs.

Figure 4 : A mature Pareto frontier approximation



Here we omit the purple dots that represent the projection of "possibly Pareto" solutions. In this case there are 4 such points, one for each of the "L" shaped bends on the green line. As some Pareto frontiers are quite dense, we don't always label these points explicitly.

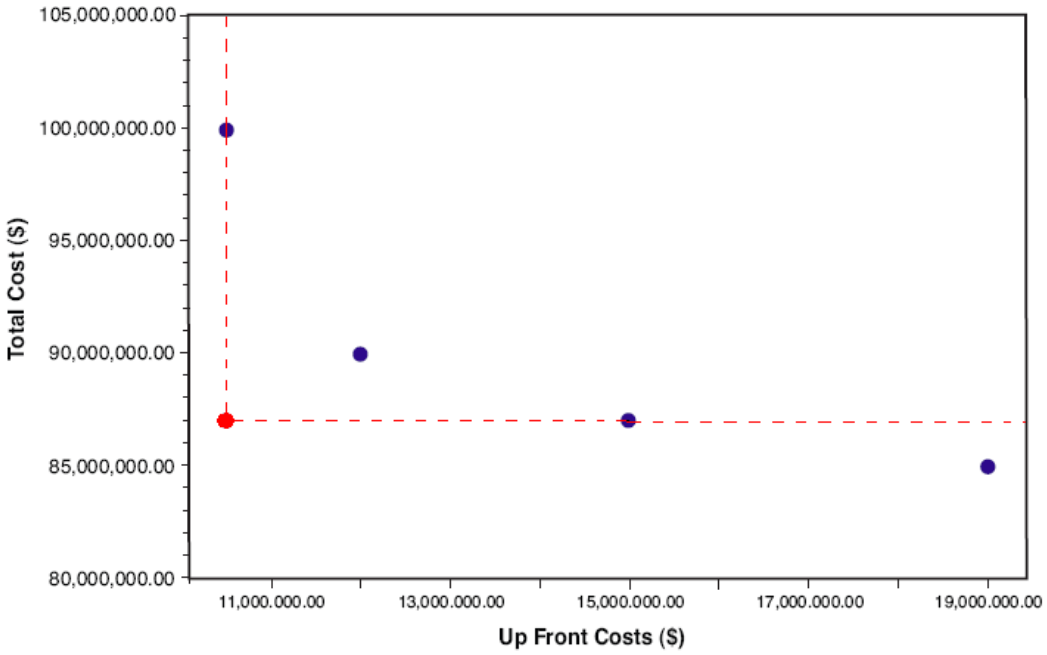
As you can see, even when the Pareto frontier consists of a small collection of points, it is worthwhile to present this result as a line graph. Users are uniformly familiar with the idea of diagramming the effect of competing goals with a trade-off curve. This curve (really, a piecewise linear function) is not only more intuitive than a scattering of points, it is mathematically consistent with the formal techniques of Pareto frontier approximation.

Approximating a Pareto frontier

Here we present a high-level description of how a Pareto frontier can be approximated.

To begin, we must define what it means for one point to "dominate" another. Specifically, one pair of (upfront cost, total cost) values will dominate another pair if it is cheaper in at least one dimension and no more expensive in the other. Below, in Figure 5, the pair plotted with the red dot dominates three of the four pairs plotted with blue dots. The red line indicates the range of points dominated by our single red dot. The blue dot at the far right is not dominated by the red dot, because the latter has a more expensive total cost.

Figure 5 : A simple lower bound plot



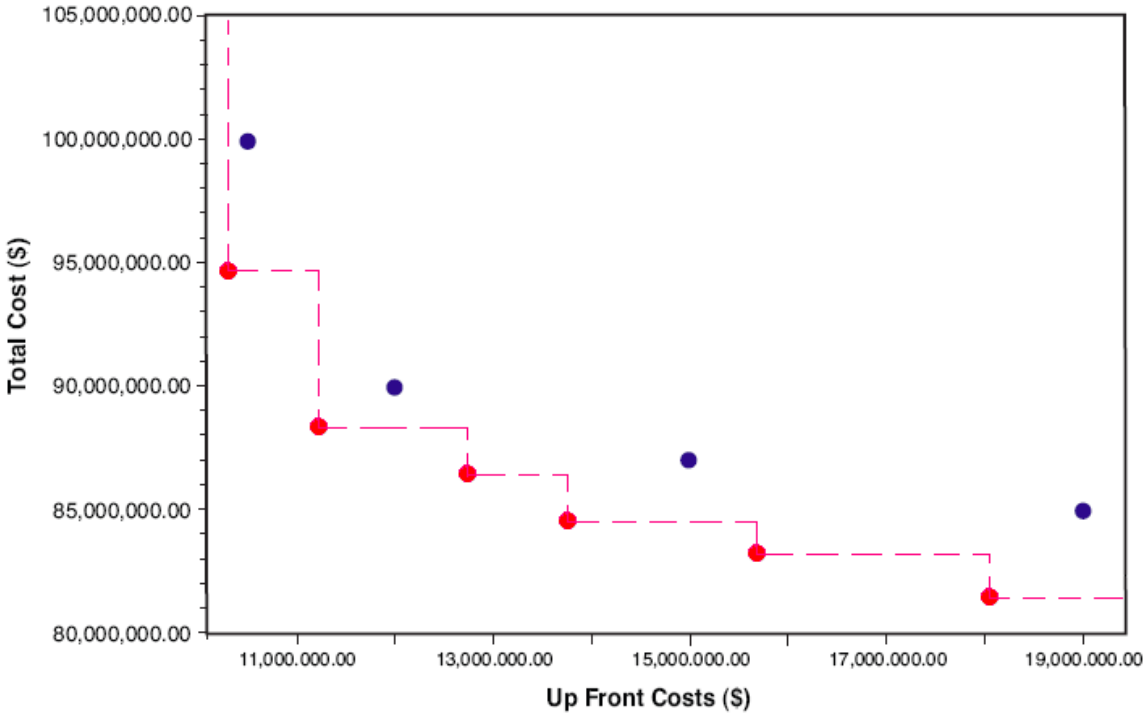
To fully approximate the Pareto frontier from beneath (or, in two dimensions, from below-and-to-the-left), we use a collection of red dots. We will use mathematical invariants to insure that our collection is exhaustive and purged of redundancies. That is to say, we will create our collection of points in a very particular way, so that we can be confident that every point on the Pareto frontier is dominated by at least one red dot. In fact, the bi-objective projection of every possible solution will similarly be dominated by at least one dot.

Moreover, every red dot is useful. If just one red dot is removed, then there will be some portion of the Pareto frontier that is no longer above-and-to-the-right of our collection of red dots. Because every red dot is useful, no red dot is redundant. That is to say, one red dot will never dominate another red dot.

We will call this set of red dots an *L* set or a lower bound set. Below, in Figure 6, we graph an *L* set that consists of 6 points. The dotted pink line indicates the limits of *L* point domination. Any point above and to the right of the pink line is dominated by one of the *L* points. As the *L* points are indomitable and exhaustive, the entire bi-objective projection of the feasibility space lies in this area (although the converse is not true – the entirety of the space dominated by *L* points might not contain the projection of feasible solutions). As part of this graph we present the same simple Pareto frontier we diagrammed earlier. The fact that this example contains more *L* points than points on the Pareto frontier is meant to illustrate the concept that the *L* set is distinct from the Pareto frontier, and exists merely to help approximate the location of the Pareto frontier. It is more likely that the Pareto frontier contains more points than does the *L*

set, or that the Pareto frontier is actually a piecewise linear function and not a collection of discrete points.

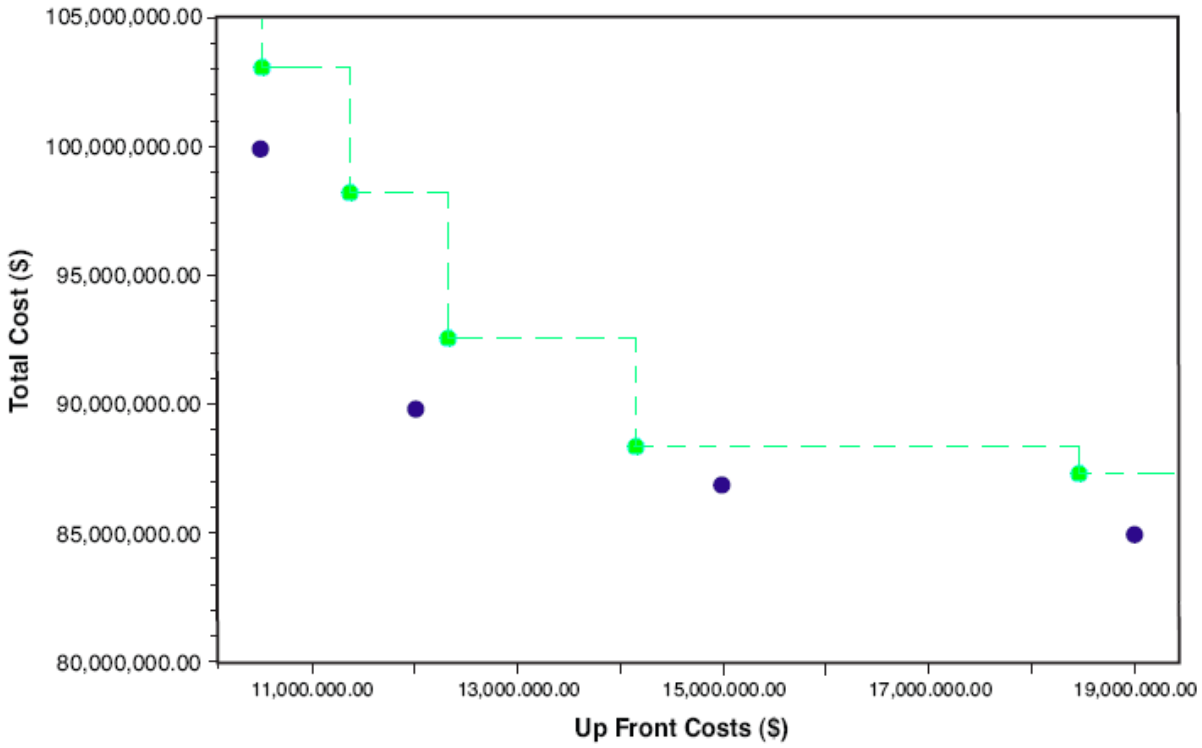
Figure 6 : An exhaustive, irredundant lower bound plot



The *L* set allows us to bound our Pareto frontier from below. The analog to the *L* set is a *U* set, which allows us to bound the Pareto frontier from above. This set (also known as an upper bound set) simply contains *achievable* points. That is to say, our *U* set contains (upfront cost, total cost) projections of known solutions. These solutions are "possibly Pareto optimal," as we haven't yet found a solution which can dominate them. We will never know if the *U* points are mapping precisely onto the Pareto frontier until the end of the algorithm, but they will keep track of our best candidates.

In Figure 7, we present the same Pareto frontier with a *U* set consisting of 5 points. In this case, any point lying above and to the right of the dotted green lines is dominated by a *U* point. Any solutions whose bi-objective projections lie in this area must not be Pareto optimal, as the *U* points are each achievable.

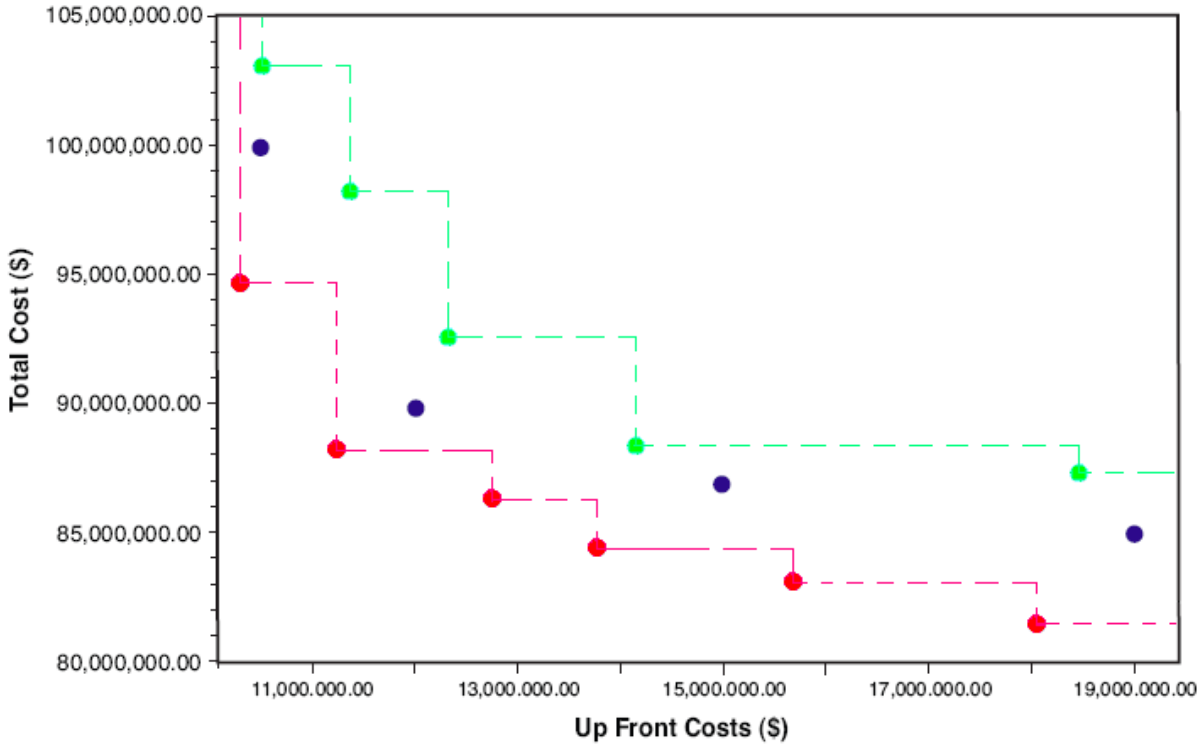
Figure 7 : An irredundant upper bound plot



A natural temptation would be marry the *L* set with a similarly comprehensive *U* set. That is, to insist that the red and green lines together create an enclosed shape, so as to insure that the Pareto frontier is entirely captured inside.

However, for technical reasons that we will discuss later, when approximating the Pareto frontier we employ a *U* set that is not necessarily comprehensive. The formal mathematics here is a little messy, but the upshot is that we won't force the green line to meet the red line, even if that means the "ends" of our graph are open. For example, in the combined graph shown in Figure 8, it is open at both ends.

Figure 8 : A full Pareto frontier Approximation



Finally, a worthwhile question to ask at this point might be "why go through all the work of maintaining two sets of points?" That is to say, why not design an algorithm that simply maintains a set of candidate *U* points, and continually improves this set until it is perfect?

There are two answers to this question. The first is that the *L* set is quite useful for almost any algorithm that we employ. It can help guide our algorithm to work efficiently, by helping us find the weakest portion of our approximation. It can also be used to let us know when to stop. If the *L* and *U* points are the exact same, then they represent an enumeration of the Pareto frontier. If not, then there is a mathematical technique for using the *L* set to assess the quality of the *U* set. This technique can be applied even if the *U* set is not comprehensive – that is to say, it can apply to "open at each end" graphs, such as Figure 8. If the *U* set represents a very near substitute for the true Pareto frontier, there might be little value in continuing. This is how the LNP V7.2 algorithm terminates.

The second reason to maintain an *L* set is that our algorithm might be interrupted before it naturally completes. Or, if running in a massively parallel environment, some of its sub-problems might fail. Regardless, we would like to provide as much information as possible to the user. Presenting a collection of "hopefully Pareto" points is obviously a much weaker partial result than presenting a pair of nearly identical lines (see Figure 4).

Assessing the quality of an optimization result

It is worthwhile to discuss the specifics of how an *L* set can be used to assess the quality of a *U* set. To begin, we briefly review how this issue is handled with single objective problems.

A user who asks LNP to find the cheapest supply chain plan with respect to a given set of constraints will often allow the tool to terminate with a sub-optimal solution. For example, a typical result might be the discovery of a solution with a total cost of $100 million that is known to be within 1% of optimality. While there might be other solutions cheaper than the one we discovered, the 1% assessment guarantees that no solution is cheaper than $99 million.

In this case, we can call the $99 million value a lower bound to the optimization problem. A patient user could always wait until the lower bound rises to precisely the same value as the total cost of the discovered solution, but that is often seen as a waste of time. After all, the input data to LNP is full of implicit predictions that are not themselves accurate to within 1%. For example, the cost of shipping is tied closely to the price of oil, and anyone who can predict the movements of the oil market to within 1% is better off playing with future contracts than designing supply chains. When the optimization accuracy of LNP is, proportionally, much better than the prediction accuracy of its input data, a solution can be thought of as being "good enough."

The fact that LNP returns a lower bound for every result means that it is a true optimization tool. The user can set a "good enough" goal based on a formal mathematical tolerance. There are other optimization tools that omit any reference to a lower bound. From a mathematical perspective, such tools are applying heuristics rather than performing true optimization. Heuristic tools can be very useful to society, but their inability to assess the quality of their solutions raise a set of troubleshooting problems that optimization tools such as LNP easily avoid.

The bi-objective optimization performed by LNP is a true optimization process that can assess the quality of its known points. However, the mathematics of how this assessment is performed is more nuanced than it is with our "1% of $100 million" example.

Using an *L* set to assess the quality of a *U* set

To begin, let's ponder what it might mean to apply a proportional tolerance to two optimization objectives. Suppose our *U* set of known points has but a single entry with total cost, upfront cost values of ($100 million, $20 million). Now imagine our bi-objective engine tells us this singleton *U* set is within 1% of the true Pareto frontier. This would imply that there is no solution that can achieve a total cost less than $99 million, nor one with an upfront cost less than $19.8 million. A solution with a total cost less than $99 million would represent a greater than 1% improvement over the span of our *U* set, even if this hypothetical solution paid a fortune in upfront expenses. Similar logic applies to a solution with a total cost less than $19.8 million.

Unfortunately, the situation becomes much more complicated when our *U* set contains more than one entry. Suppose we have a *U* set with two points – ($100 million, $20 million) and ($200 million, $10 million). What does it mean to say this two entry set is within 1% of the true Pareto frontier? The most natural definition is that for each *U* point, there is no possible solution that improves one dimension by more than 1% without degrading the other objective. The easiest observation we can make is that no solution will provide more than a 1% improvement on the cheapest total cost value in our *U* set. That is to say no solution has a total cost less than $99 million, regardless of its upfront cost. Similar logic dictates that no solution will have an upfront cost less than $9.9 million.

However, the idea that our two element *U* set is within 1% of perfection has other, less obvious implications. We can also conclude that no solution with an upfront cost of $10 million or less can achieve a total cost better than $198 million, and similarly no solution with a total cost of $100 million or less can achieve an upfront cost better than $19.8 million.

This idea of our *U* set being proportionally close to the Pareto frontier is quickly becoming unwieldy. One problem is that the logic of considering how much one objective can be improved without obligating a penalty in the other objective results in a tangle of implied restrictions. In one dimension, it's easy to deduce where the optimality limits are relative to our single known point. In two dimensions, it's not at all obvious.

Even worse, however, is that thinking in terms of proportions gives us an inconsistent set of tolerances. We are willing to allow a $2 million improvement in total cost to our ($200 million, $10 million) point, but we can abide by no more than a $1 million total cost improvement to our ($100 million, $20 million) solution. Such a double standard seems capricious. Even worse, it will apply a visually inconsistent standard to our red-green approximation graph.

The LNP approach to measuring *L,U* fitness reverses both of these assumptions. First, since we know the *L* set collectively limits the entire Pareto frontier, we will only assess whether or not

the *U* set is close enough to the *L* set. Second, we will apply a uniform (total cost, upfront cost) tolerance for the entire graph.

For example, if the LNP bi-objective code tells us that our *L,U* set has achieved a ($1 million, $0.1 million) tolerance, then we can conclude that for each *L* point, there must exist at least one *U* point whose total cost is within $1 million, and whose upfront cost is within $0.1 million. Since each Pareto point is dominated by at least one *L* point, this same logic can be applied to the Pareto points themselves. That is to say, for each (as yet not fully determined) Pareto point, we know that there is at least one discovered *U* point with a substitution penalty no worse than ($1 million, $0.1 million).

This standard is fairly easy to understand, but it does require us to reorient our thinking. Rather than ask "by how large a percentage can we possibly improve our known solution?", we ask "what sort of penalty is incurred by substituting our known solutions for the true Pareto frontier?" It would be impossible to compute this penalty as a proportion of the (as yet unknown) best possible solutions. Moreover, a uniform tolerance for each objective makes good sense for a graphical result. Hence, we measure this penalty in absolute numbers and not percentages.

A careful reader would observe that we have elided the computational details for setting the ($1 million, $0.1 million) tolerance vector itself. It is perhaps an annoying property of Pareto frontier approximation that the same *L,U* set can be said to meet completely inconsistent tolerances. That is to say, when LNP achieves ($1 million, $0.1 million) accuracy, it might also accidentally achieve ($5 million, $0.03 million) accuracy. Since there is no unique tolerance vector associated with a given *L,U* set, LNP must apply judgment to determine which tolerance vector to target. This computation is done very early in the approximation algorithm through a heuristic technique that considers both the overall Pareto span and the magnitude of the largest Pareto objective values.

Although the specific optimization goal is set using a heuristic, the bi-objective optimization itself is applying a rigorous standard of fitness that is consistent with true mathematical optimization. As with single objective optimization, a sufficiently patient user could force the optimization process to bring the two-dimensional lower bound into (near) perfect alignment with the known points, if such precision were deemed important.

Initializing the *L,U* sets

It is a relatively straightforward matter to begin the LNP bi-objective algorithm with a valid *L,U* set initialization. A simple example should suffice to illustrate the process.

Suppose we solve the optimization problem that minimizes total cost irrespective of upfront cost to within a single objective optimization tolerance of 1%. Similarly, let us solve the problem that minimizes upfront cost irrespective of total cost to within the same tolerance. If the respective results are $100 million total cost (with an upfront cost of $20 million) and $10 million upfront cost (with a total cost of $200 million) then we can initialize our *L,U* sets as follows.

The *U* set can consist simply of the two discovered points. That is, we initialize *U* to be {($100 million, $20 million), ($200 million, $10 million)}. The *L* set can be initialized by the two *lower bounds* from each anchoring problem. That is to say, we initialize the *L* set to be {($99 million, $9.9 million)}. Remember, our invariant is that every possible solution must be dominated by at least one point in the *L* set. Since the single *L* point we have constructed combines the Candide-like possibilities from both single objective problems, our invariant is true by definition.

A careful reader might suspect that this *L* set initialization is simply postponing a computational headache. It is easy enough to construct a single *L* point that dominates the entire range of possibilities, but how does one maintain and update a collectively exhaustive *L* set containing multiple incomparable entries? The answer lies in the discovery of bounding points –points that cannot be strongly dominated by any feasible solution. We explore these details in the following sections.

Refining the *U* set

Before we consider how bounding points can refine an *L* set, let us first tackle the easier topic of improving the *U* set of Pareto candidates. The *U* set is refined by making it “cheaper.” That is to say, the *U* set is improved by increasing the portion of the graph that is above and to the right of at least one *U* point. This is accomplished by discovering a feasible projection that is not dominated by any of the current points in *U*, and adding this new Pareto candidate to the *U* set. Since we are allowing non-Pareto points to have membership in the *U* set, this new entry may dominate points already in *U.* Such points are easily identified and removed.

We graph these two possibilities with Figures 9 and 10. Figure 9 introduces a worthwhile discovered point which is not comparable to any elements already in *U*. Figure 10 demonstrates how a new discovered point might dominate some points already in *U*, and thus those points would need to be removed from *U* as part of the refinement process. We include the *L* set in both graphs, to provide some reference to the limits of where worthwhile new discovered points might appear.

Figure 9 : A simple *U* set insertion

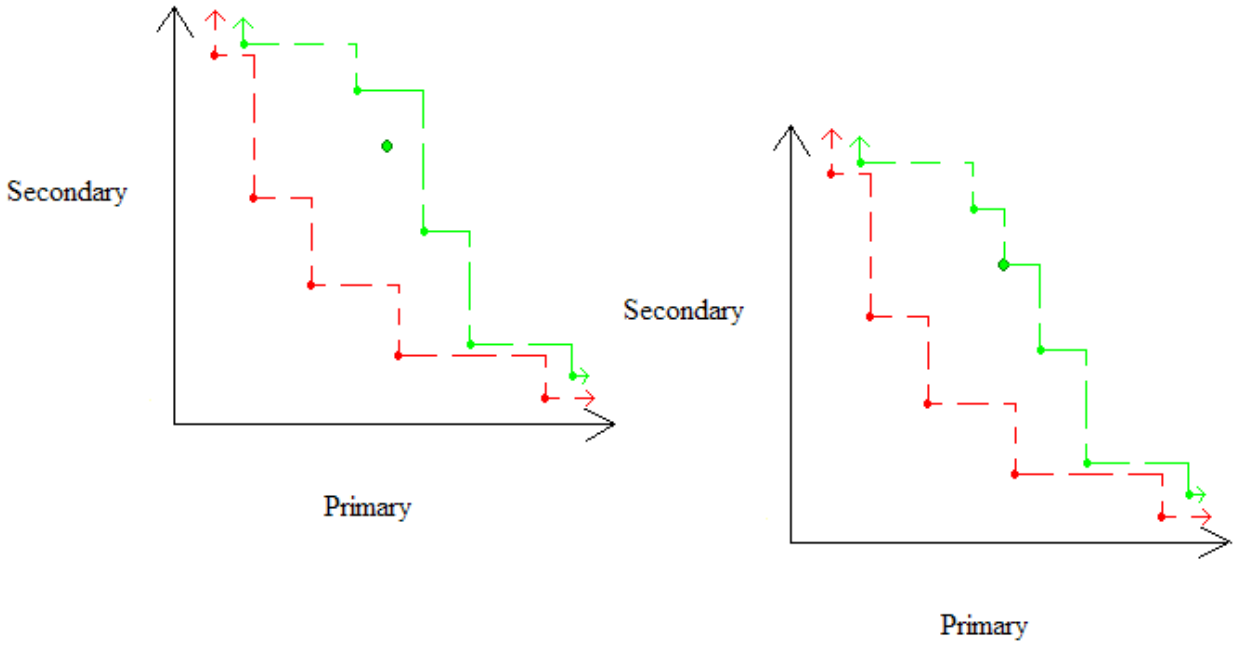
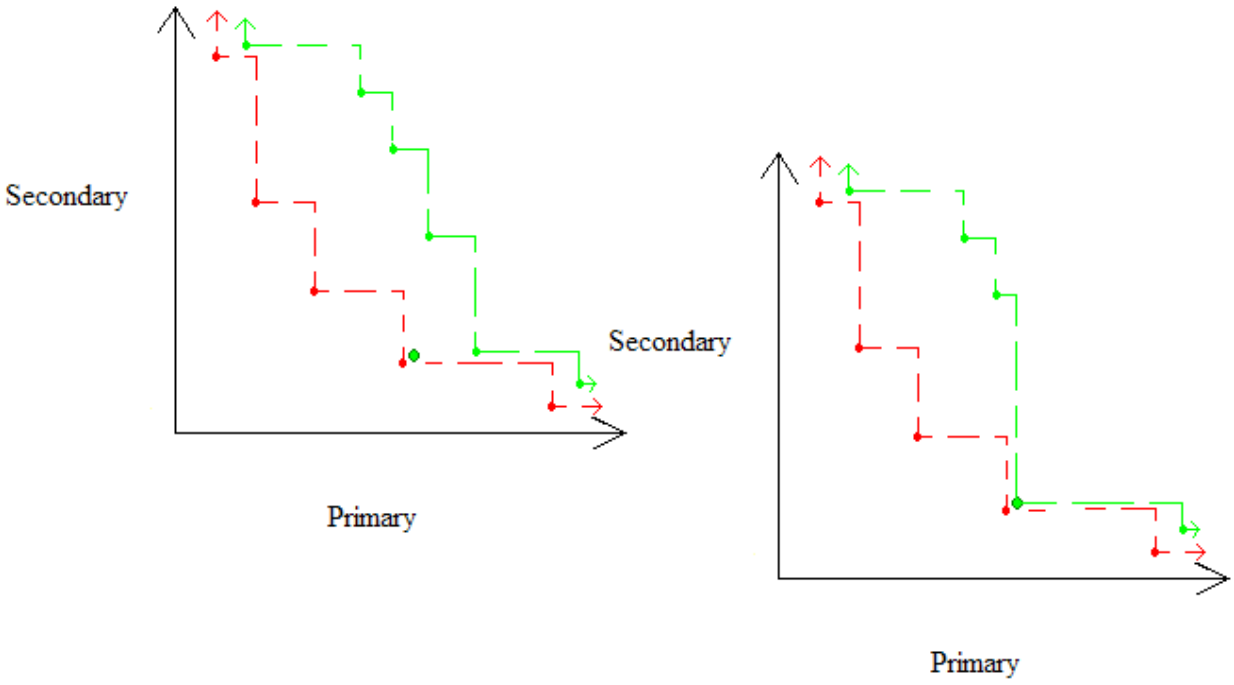


Figure 10 : A *U* set refinement



Refining the L set

Refining the *L* set is somewhat more complicated. We refine the *L* set by discovering that points already in *L* are actually unattainable, removing these points, and replacing them with two potentially attainable points that dominate a smaller projection of the objective space. Although the bookkeeping involved here isn't terribly complicated, it's interesting enough to warrant a detailed explanation.

Suppose that, as a by-product of solving a sub-problem, we are able to derive a limit to optimality. That is to say, suppose we discover that there is no solution that can doubly improve on the (total cost, upfront cost) values of ($150 million, $15 million). We shall call such a point a *bounding* point, and we shall see below that they can be discovered quite naturally.

Intuitively, the discovery of our new bounding point means that the singleton {($99 million, $9.9 million)} *L* set can be improved upon. Specifically, imagine a point that is dominated by ($99 million, $9.9 million) and not dominated by either ($99 million, $15 million) or ($150 million, $9.9 million). This can only happen if this point is also strongly dominating our bounding point of ($150 million, $15 million). Since the definition of a bounding point is that which cannot be strongly dominated, this means the set of feasible points dominated by ($99 million, $9.9 million) must also be dominated by either ($99 million, $15 million) or ($150 million, $9.9

million). Thus, our bounding point of ($150 million, $15 million) allows us to transform our *L* set from the simple {($99 million, $9.9 million)} to {($99 million, $15 million), ($150 million, $9.9 million)}.

This logic is surprisingly easy to extend both to larger pre-refinement *L* sets and to a 3 (or more) dimensional objective space. Although *L* points are critical to assessing the quality of a *U* set, bounding points are the building blocks used to improve an *L* set. Once we discover the Candide-like point that dominates everything, we can then initialize a singleton *L* set and refine it with whatever bounding points might emerge from our sub-problem results. These bounding points can be accepted in any order. If a bounding point is somehow "lost in translation," the only harm that results is an *L* set that is somewhat less refined. This ability of both *U* sets and *L* sets to be developed in a robust manner is critical to the "embarrassingly parallel" status of bi-objective optimization. Single objective optimization doesn't lend itself so naturally to the weaving of an aggregate result from a collection of simple sub-results.

As an aside, some multi-objective algorithms consider points that cannot be achieved as their *L* set building blocks. The logic of incorporating "unsat" points is very similar to the bounding point refinement we describe above. This is useful to remember, as some numerical engines which do not provide true lower bound results can be used to identify models which cannot be solved, and unsat points can generally be derived from provably infeasible models.

We briefly generalize the refinement process described above.

1. Let us use *b* to refer to the newly discovered bounding point.
2. Identify $\hat{L}$, which is the sub-set of *L* that dominates *b.*
3. Replace each point $\hat{l} \in \hat{L}$ with the refined points $(b_p, \hat{l}_s), (\hat{l}_p, b_s)$
4. Remove the redundancies from $\hat{L}$ that have been created by this refinement of the individual points. I.e. if your refinement creates $\hat{l}'$ and $\hat{l}''$, such that $\hat{l}''$ dominates $\hat{l}'$, then remove $\hat{l}'$.

Step 4 is needed when $\hat{L}$ contains more than one entry. Step 3 changes the space dominated by *L* while insuring that this space contains all the feasible projections. Step 4 removes redundancies from *L* without changing its realm of domination.

We briefly illustrate this process with Figures 11 and 12 below.

Figure 11 : The *L* set pre-refinement logic

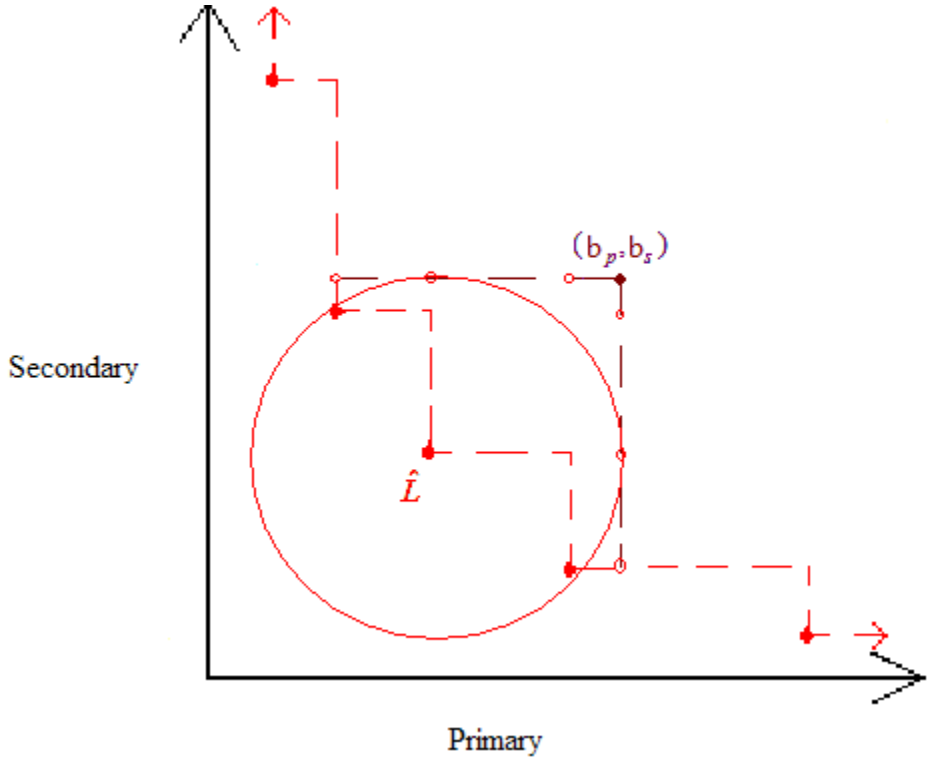
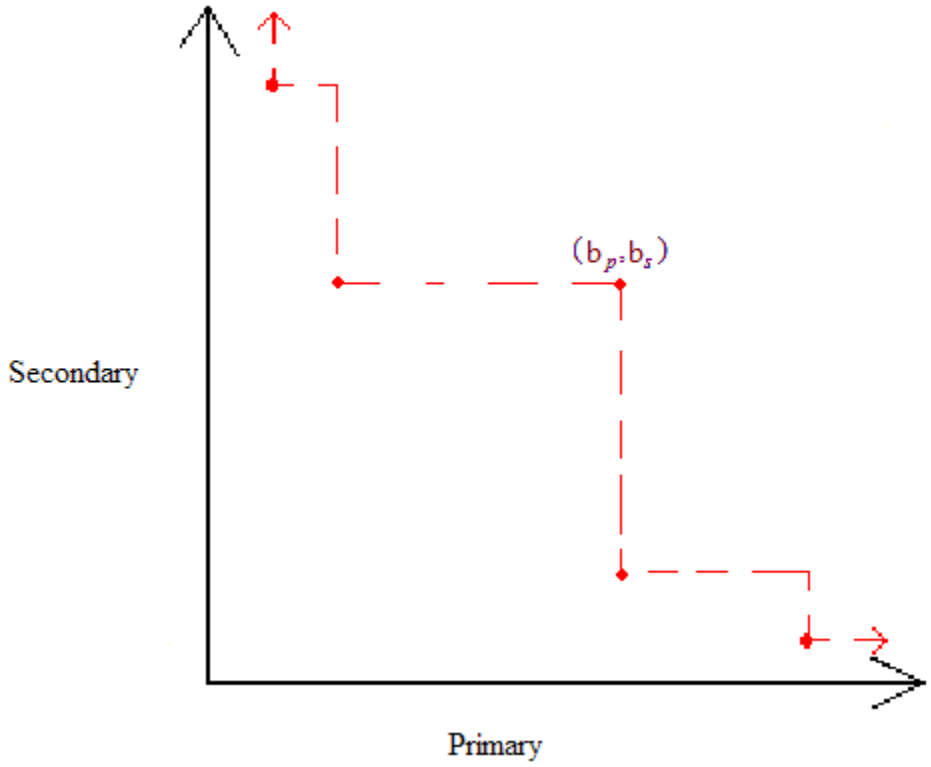


Figure 12 : The *L* set after refinement

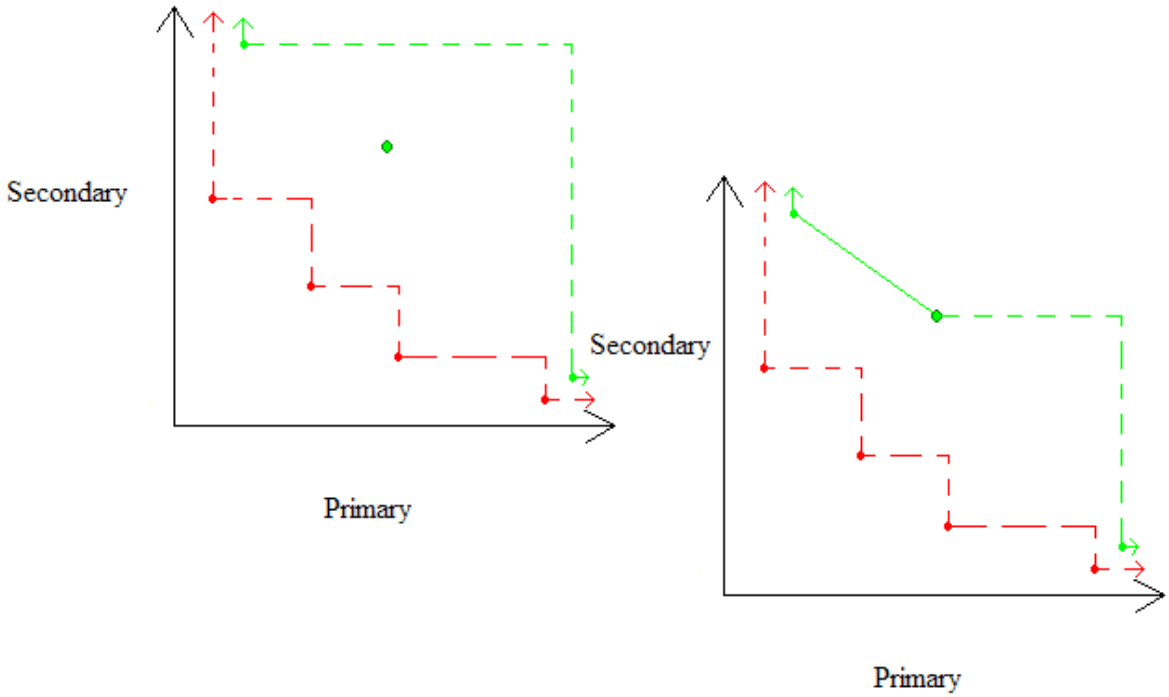


Aside – line segments

Although the LNP research and development was based around the creation of individual *L* and *U* points, there are mathematical techniques for creating line segments that can approximate the Pareto frontier. Although these methods are typically associated with pure linear programming (that is to say, with models that have no Boolean or integral variables) they represent powerful tools for recognizing certain types of mixed integer programming Pareto frontiers. In particular, some models have a relatively small number of structural Boolean variables whose values exert control over a large body of continuos variables. Such models could easily generate Pareto frontiers that consist of a series of downward sloping line segments.

In Figure 13, we illustrate a *U* set update where the newly discovered achievable point has the same integer variable profile as one of the two pre-existing *U* points. In this case, every convex combination of these two *U* points is a *U* point as well, and thus the *U* set update involves connecting the two points with a line segment. We draw a solid line segment here to indicate that every solution whose bi-objective projection maps along this segment is itself a Pareto candidate. The dashed green lines, by contrast, indicate where the Pareto candidates are not. The distinction here is the same as the difference between open and closed intervals.

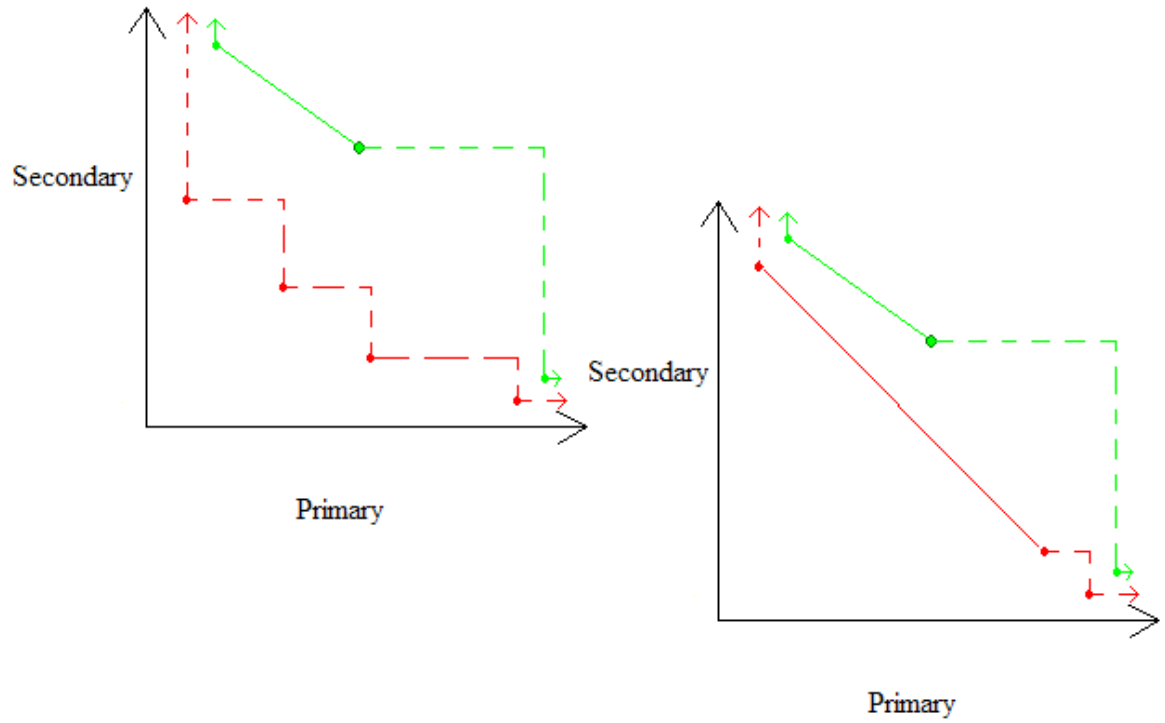
Figure 13 – A *U* segment update



It is also possible to create *L* segments. In this case, the entire segment can be thought of as consisting of both bounding points and *L* points. These segments are created when solving blended objective sub-models. For example, if the objective of *minimize primary + secondary* has a *lower bound* result of 1, then it means the entire line associated with *secondary = 1 – primary* can be thought of as a continous bounding segment.

We illustrate this type of update in Figure 14. Here we use a solid red line to indicate a line segment that consists of both bounding points and *L* points.

Figure 14 – A bounding segment update to the *L* frontier



Although *L* and *U* segments are somewhat less intuitive than *L* and *U* points, they will surely be needed to create a truly industrial Pareto frontier optimization engine. The bookkeeping associated with an arbitrary mix of segments and points is particularly messy and beyond the scope of this paper.

Single objective sub-problems

As one might expect, the general technique for solving a bi-objective optimization problem is to utilize the power of single objective optimization technology. A bi-objective algorithm will formulate single objective sub-problems in such a way as to guarantee that each sub-problem result improves the *L,U* approximation graph. That is to say, a given sub-problem will either generate a bounding point which is strongly dominated by at least one of the current *L* points, or it will discover an achievable point which is not dominated by any of the current *U* points.

We've already seen two examples of single objective sub-problems in the **Initializing the *L,U* sets** section. To begin our approximation, we employed the simplest of all single-objective sub-

problems, those which minimize one objective while ignoring the other. The extraction of achievable points from our two anchoring problems was fairly straightforward. Although the relationship between the initial Candide-like *L* point and the two sub-problems was explained colloquially, there is a formal mathematical proof based on bounding points that justifies our *L* set initialization.

Of course, the bi-objective Pareto frontier can not be approximated without solving single objective sub-problems that pay attention, in some fashion, to both objectives. In this paper we explore two types of sub-problems – those which constrain one objective while minimizing the other, and those which diagnolize the bi-objective space by reframing both objectives as a single Chebyshev distance.

Constraint based sub-problems

Constraint based techniques are the easiest to consider and implement. While Chebyshev sub-problems are typically needed to perfectly enumerate a bi-objective Pareto frontier, constraint based sub-problems tend to present fewer implementation hurdles, particularly when grafting bi-objective functionality onto a pre-existing, single objective application. As a result, the LNP V7.2 algorithm relies exclusively on the clever application of secondary objective constraints.

In addition, a massively parallel compute resource would surely avail itself of constraint based sub-problems to quickly jump-start a high quality Pareto approximation. We demonstrate this by revisiting the Pareto frontier first seen in Figure 1. Imagine if 10 different single-objective problems were solved in parallel – one placing no restrictions on up-front costs, and the other 9 applying up-front cost restrictions at $1 million intervals between $11 million and $19 million, inclusive. The *L,U* set that would result from these 10 sub-problems is demonstrated in Figure 15.

Figure 15 : A tight approximation from 10 independent sub-problems

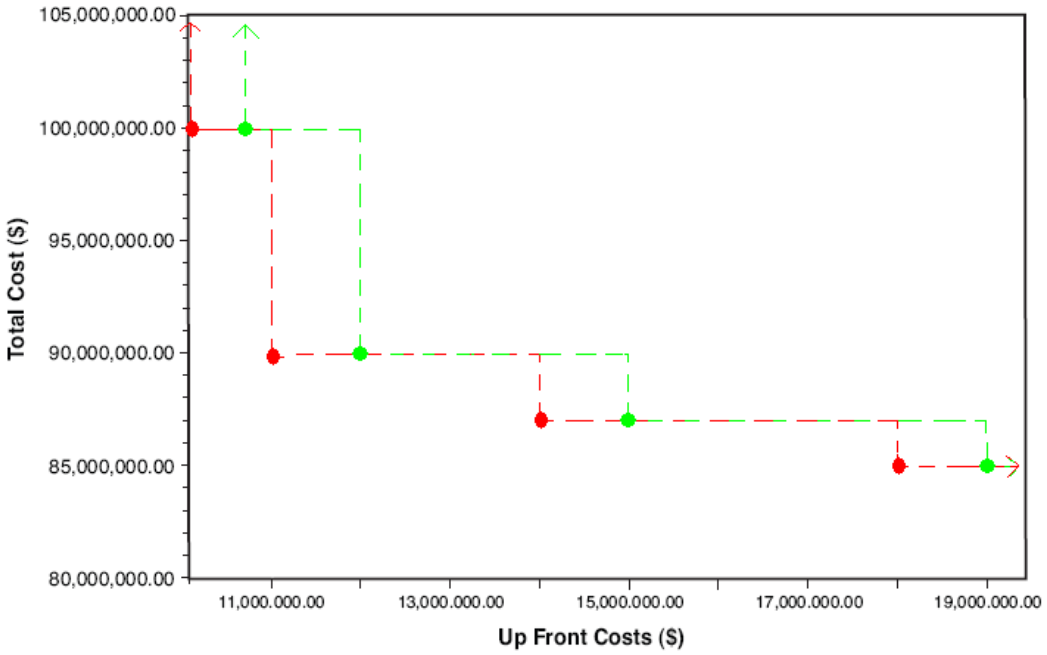


Figure 15 shows our brute-force approach discovering 4 different solutions. If you compare Figure 1 to Figure 15, you can see that 3 of the 4 green points are actually on the Pareto frontier. The solution with a total cost of $100,000,000 and an upfront cost of $10,900,000 is not Pareto, since Figure 1 shows that there is a Pareto solution with the same total cost and a smaller upfront cost.

However, Figure 15 doesn't allow us to recognize that the other 3 discovered points are in fact Pareto. Any algorithm based on *L* and *U* sets will recognize a Pareto point only by discovering that a *U* point is equivalent to an *L* point. Thus, Figure 15 can only serve as an approximation to the Pareto frontier. As a rule, this is true of any *L,U* set that is generated by constraint based sub-problems. We discuss how a high quality approximation can be upgraded to a perfect Pareto enumeration in the **Chebyshev sub-problems and bi-objective enumeration** section. In this case, our approximation can be perfected by a second pass of 4 parallelized sub-problems, 3 of which applying a diagonalized Chebyshev objective.

Using constraint based sub-problems to refine the *L* and *U* sets

Suppose we solve the optimization problem that minimizes total cost without spending more than $15 million in upfront costs. Let's assume the result of this sub-problem is the discovery of a solution with a (total cost, upfront cost) profile of ($151.5 million, $14 million), while achieving an optimization tolerance of 1%. This sub-problem result can potentially refine both our *U* set and our *L* set.

The former update is fairly straightforward. We know now that ($151.5 million, $14 million) is a feasible projection. If this point is not dominated by any elements of our *U* set, we can apply the steps described in **Refining the *U* set** to allow our set of known points to more closely resemble the true Pareto frontier.

The latter update is a bit more complex. Although it might not be immediately obvious, our sub-problem result has told us that ($150 million, $15 million) is a valid bounding point. To see why this must be true, consider the alternative. Suppose there is a supply chain plan with a bi-objective projection of ($149.99 million, $14.99 million). This hypothetical solution would be feasible in the universe of our restricted, single-objective sub-problem (that is, it has an upfront cost less than $15 million). Moreover, the total cost of $149.9 million is not only superior to the $151.5 million best known value of our sub-problem, it is better than the *best possible* restriction implied by our optimization tolerance of 1% (that is, $149.99 million is less than 99% of $151.5 million = $150 million). Thus, our hypothetical plan of ($149.99 million, $14.99 million) must not be possible, and neither is any plan whose bi-objective projection strongly dominates ($150 million, $15 million). This lets us apply the bounding point ($150 million, $15 million) when exercising the steps described in **Refining the *L* set.**

The LNP bi-objective algorithm can efficiently approximate the Pareto frontier by appropriately setting both of these sub-problem parameters. That is to say, LNP will set both the upfront cost budget and the sub-problem optimization tolerance in a manner that is sensitive to the maturity of the approximation.

Initially, LNP will apply upfront cost restrictions that are not particularly close to actual *L* set values, and will allow sub-problems to terminate long before they have found a truly optimal solution. When the gaps between the *L* and the *U* points are quite large, there is little advantage in finding bounding points that are close to *L* points, nor in finding new achievable points that are close to current *U* points.

As the approximation matures, LNP will recognize that the *L,U* approximation set is achieving compliance with ever more stringent (total cost, upfront cost) tolerance vectors. This, in turn, will justify upfront cost restrictions that are closer to *L* set values, and sub-problem results with smaller optimization tolerances. Although this technique is unlikely to ever truly enumerate the Pareto frontier, it is an efficient approach for creating a high quality Pareto frontier approximation.

Chebyshev sub-problems and bi-objective enumeration of a complete Pareto frontier

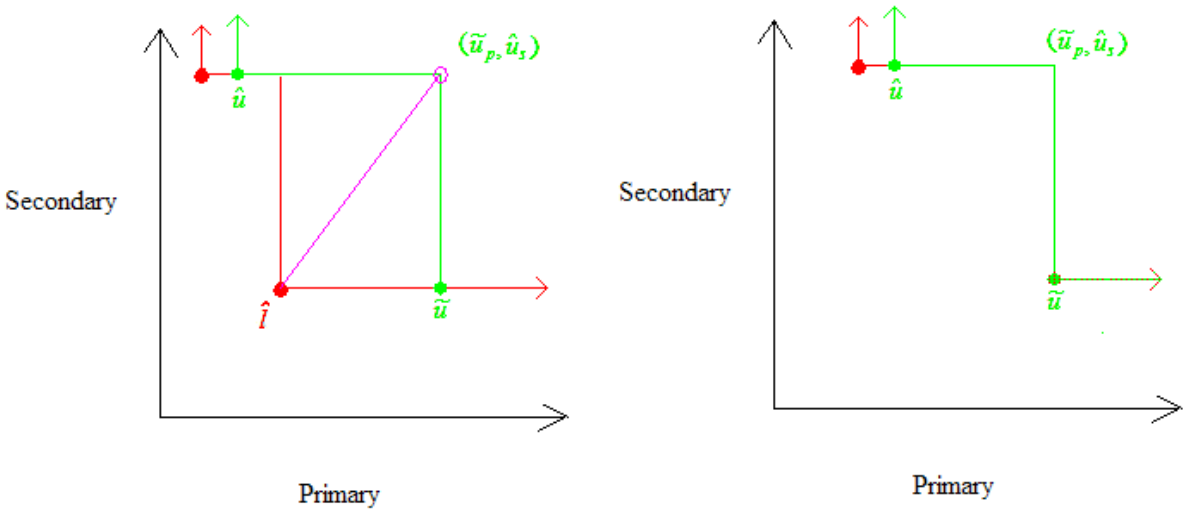
Chebyshev techniques implement a Chebyshev distance as the goal of the single objective sub-problems. A Chebyshev distance is simply the “worst-of-both-worlds” distance measurement when considering the restrictions of a rectilinear grid. For example, if ones apartment was 10

blocks south and 4 blocks west from work, then your morning commute would be 14 blocks when measured by a pedometer, but only 10 blocks when measured by a Chebyshev distance. If you moved 2 blocks west so as to reduce your total commute to 12 blocks, the Chebyshev distance would remain the same.

The principle advantage of weaving a Chebyshev distance into our sub-problems is to allow for a fortunate sub-problem result to fully explore a rectangle in our *L,U* graph. A Chebyshev objective that fails to improve the *U* set will be guaranteed to radically improve our *L* set, to the extent that it will create a bounding point that completely validates a swath of otherwise unexplored space.

This event is illustrated in Figure 16. Here, we draw a purple line to connect an *L* point with a worst-of-both-worlds extrapolation of two adjacent *U* points. The discovered bounding point will always appear along this diagonal. In this case, there are no solutions which can improve upon the appropriately scaled Chebyshev distance from the *L* point, and thus the bounding point falls on the outermost end of the purple line. When such a bounding point is applied in **Refining the L set,** the unexplored rectangle dominated by the *L* point is completely removed.

Figure 16 : A Chebyshev generated bounding point removes an entire rectangle

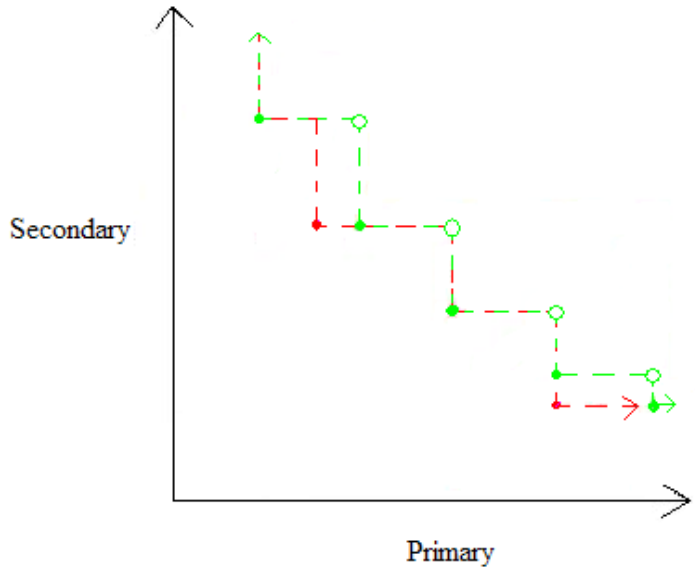


If enough of these updates occur, our *L* set will perfectly overlap with our *U* set, and the Pareto frontier will be completely enumerated. Such a fortunate series of events can't be assured for every bi-objective MIP. However, we can guarantee that if the Pareto frontier consists of a discrete collection of points, then the number of sub-problems that needs to be solved will be a relatively small scalar of the cardinality of the Pareto frontier. That is to say, if there are *P* points

in the Pareto frontier, then simply by solving Chebyshev sub-problems and updating our *L* and *U* sets, we will enumerate the full Pareto collection with no more than *4P+1* sub-problems. Even better, it is unlikely that we will need much more than *3P* sub-problems, and we might use as few as *3P/2* sub-problems (although this best case is just as unlikely as the *4P+1* worst case).

The precise mathematics of Chebyshev based bi-objective enumeration is beyond the scope of this text, but we can give a high level overview of the associated algorithm. To begin, we must introduce one more flavor of bi-objective point – the *nadir* point. A nadir point simply represents the "worst of both worlds" of two adjacent achievable points. In Figure 17, we produce an *L,U* approximation graph with 4 nadir points, each identified by hollow green dots. This graph represents a partial enumeration of the Pareto frontier, as two of our *U* points perfectly overlap with *L* points, and three others are weakly dominated by exactly one *L* point.

Figure 17 : A partial enumeration with four nadir points



The nadir points are useful for orienting the Chebyshev objective, as each *L* point which is not equivalent to a *U* point must strongly dominate a nadir point. In fact, we can exploit the converse of this observation, which is that if there are no *L* points which strongly dominate a nadir point, then the *L* and *U* points must be equivalent. Thus our Chebyshev enumeration algorithm is quite simple:

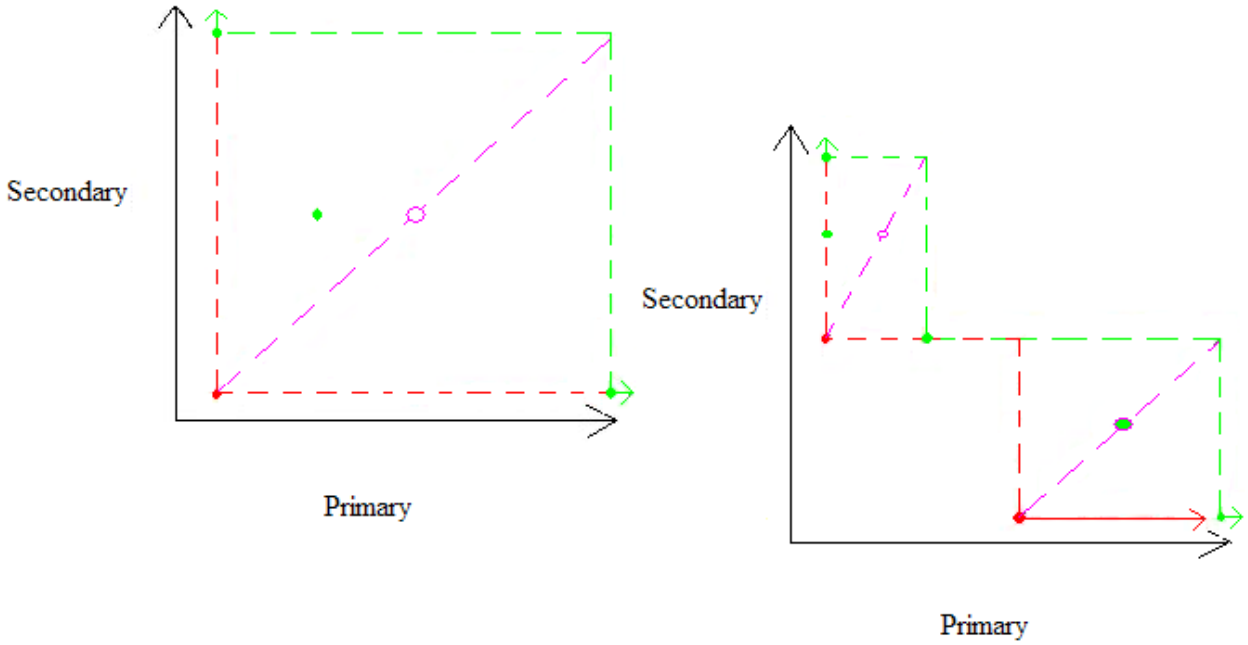
1. If there are no *L* points which strongly dominate a nadir point, then the Pareto frontier has been enumerated.
2. Otherwise, find an *L* point which strongly dominates a nadir point.

3. Solve the single objective optimization problem which minimizes the Chebyshev distance from the *L* point, with the distance measurements scaled relative to the nadir point.
4. Use the bounding and achievable points from this sub-problem to update the *L,U* sets.
5. Return to step 1.

As an aside, we should note that while this algorithm is well defined for more than two objectives, the essential condition of step 1 will not hold. That is to say, in 3 or more dimensions, you can create *L,U* approximation sets such that there are no *L* points which strongly dominate any nadir point, and yet the *L* and *U* sets will fail to achieve perfect alignment. Thus, enumerating a Pareto frontier for more than 2 dimensions is likely to be much more complex.

We illustrate this algorithm with a series of figures. Figures 18 through 20 illustrate how the Chebyshev sub-problem results move us closer to a perfect enumeration of the Pareto frontier. In each graph (except for Figure 20) we indicate which pair of *L* and nadir points has been selected, and where the resulting bounding and achievable points will appear.

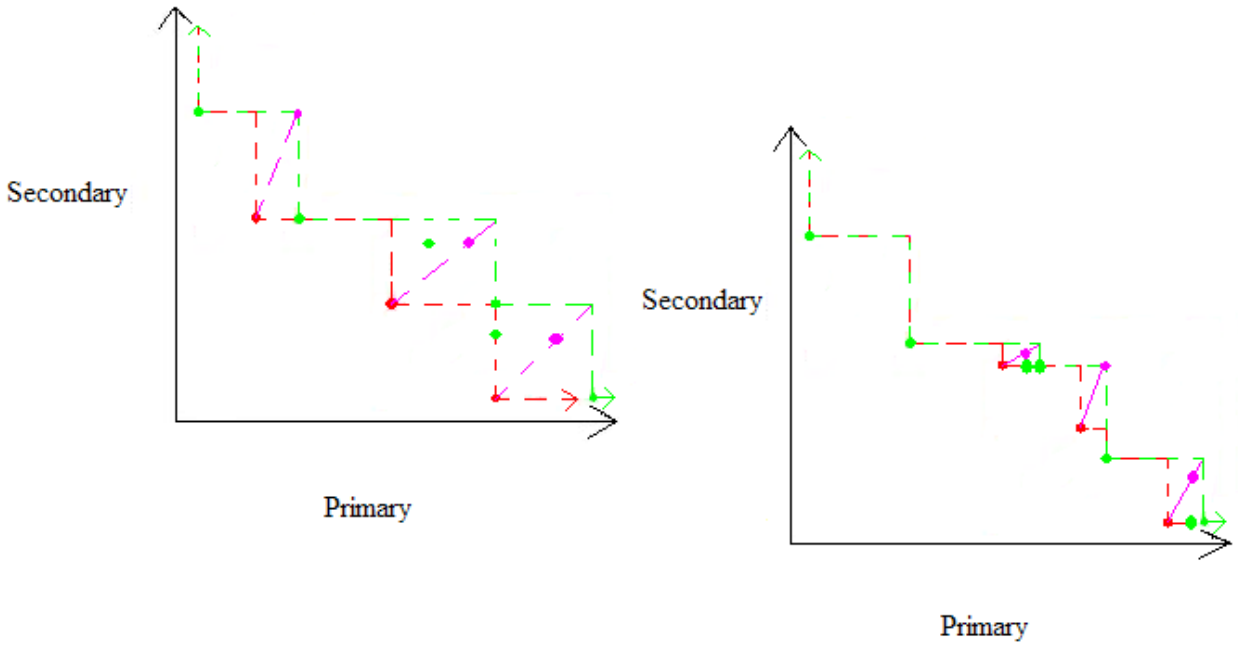
Figure 18 : The first three sub-problems



In Figure 18, we see the solution and update from the initial Chebyshev sub-problem, as well as the solution from the next two sub-problems solved in parallel. Note that the bounding point, shown in purple, will always appear somewhere along the purple diagonal connecting a nadir

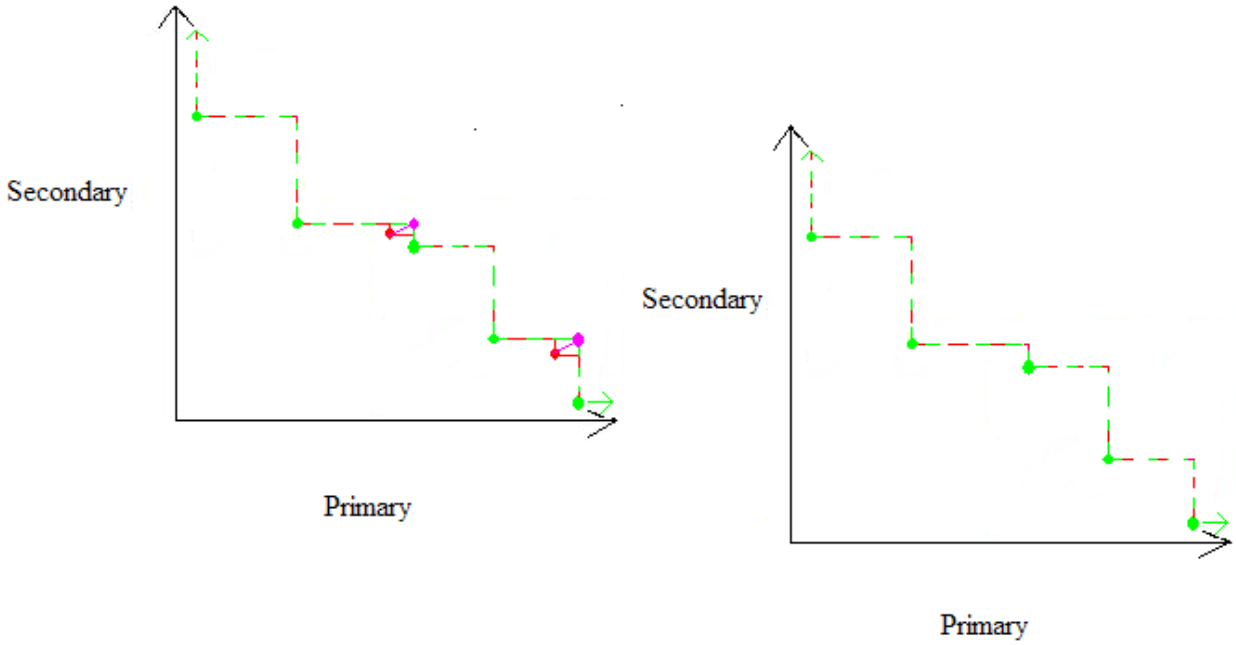
point with the *L* point that strongly dominates it. The newly discovered *U* point will either weakly dominate this bounding point or be equivalent to it. This new *U* point might weakly dominate a pre-existing *U* point, or it might be completely inside the nadir-ideal rectangle that determines the Chebyshev problem. Note that the former result will necessitate the replacement of a pre-existing *U* point. Remember, our algorithm doesn't require the *U* points to be truly Pareto, as such a "polishing" requirement would inhibit the rapid decomposition of our problem into parallelizable sub-problems. Rather, we recognize a Pareto point naturally, when a *U* point falls into alignment with an *L* point.

Figure 19 : Three computational threads working simultaneously



In Figure 19 we see how 3 computational threads working in parallel can efficiently address the Chebyshev enumeration process. This graph also provides another demonstration of a red-green rectangle that is fully explored by a Chebyshev result. That is to say, when there is no solution whose oriented Chebyshev distance from the ideal point is better than that of the nadir point, then the bounding point aligns with the nadir and the entire rectangle is removed. This is one of the two ways in which a Pareto point is recognized – the other being when no solution is interior to the ideal-nadir rectangle but there is an improving solution along an edge. The proof for this algorithm works through each of these cases in detail and demonstrates that measurable progress is made in each situation.

Figure 20 : An algorithm terminates



In Figure 20, we see how the Chebyshev algorithm terminates. When there are no nadir points that are strongly dominated by an ideal point, we can rest assured that the *L* and *U* points must be in alignment. Figure 20 also demonstrates the wisdom of maintaining *L* points as an intermediate result. Before the final two sub-problem updates are performed, we can not only demonstrate that 5 Pareto points have been discovered but, more importantly, that any undiscovered Pareto points will land only within two small rectangles. An algorithm that is terminated at this stage presents a result that is a near perfect substitute for the final, fully explored graph.

In this example, our Chebyshev algorithm required 13 sub-problems (including the two initialization problems) to discover 5 Pareto points. This is a common result. While our mathematical proof implies a potential worst case of *4P+1* one-objective sub-problems, numerical testing indicates that the likely total will be between *2P* and *3P*.

A careful reader will note that this algorithm will enumerate the Pareto frontier only if it applies a zero percent optimization tolerance to each of the one objective sub-problems (to include the two initialization problems). While true, this concern is more of a comment on the challenges of single objective optimization than it is on the tractability of bi-objective Pareto enumeration. That said, a modeler who is unsure of the difficulty of his single objective sub-problems could simply apply a hybrid, “first approximate then enumerate”, algorithm. We outline this approach as follows.

- Solve the two initialization sub-problems to within a small, but non-zero, optimization tolerance.
- Derive a tolerance vector from the two resulting optimization gaps.
- Repeatedly solve a series of Chebyshev sub-problems to within a non-zero optimization tolerance. Stop only when the resulting *L,U* sets meet the requirements of the tolerance vector.
- Re-solve the two anchor problems with a zero percent optimization tolerances.
- Repeatedly solve a series of Chebyshev sub-problems with a zero percent optimization tolerance. Stop when there are no ideal-nadir rectangles to collapse. This will occur only when the *L,U* sets are in perfect alignment and the Pareto frontier has been enumerated.

This approximate-enumerate hybrid algorithm does not lead itself to a mathematical proof limiting the number of required sub-problems. However, since we have such a proof for the pure enumeration algorithm, it seems reasonable to conclude that the hybrid algorithm won't require an unreasonable number of sub-problems when the Pareto frontier is countable. Moreover, the approximate-enumerate hybrid would be useful for models which may or may not have a countable Pareto frontier. Particularly when super-charged by a highly parallelized compute resource, such an algorithm could be relied upon to efficiently approximate all bi-objective Pareto frontiers, and enumerate those frontiers for which such a result is realistic.

Summary and Conclusion

The bi-objective Pareto frontier is particularly well positioned to exploit the modern optimization landscape. The efficiency of cutting-edge single-objective engines, the ubiquity of multi-core processors, and the inevitability of cheap, scalable cloud cloud compute resources all work in favor of an optimization problem that naturally decomposes into a manageable number of independent single-objective sub-problems. In addition, as the business community hungers for big-data analytics, it is natural that they will be receptive to a graphical result presented as a classic Cartesian plot.

The LNP V7.2 bi-objective functionality was warmly embraced by the strategic network design community. This paper has hopefully demonstrated how similar enthusiasm can be realized by applying this functionality to the optimization market at large.

Resources

- For more information on Supply Chain Planning, and how it relates to bi-objective optimization, see *Supply Chain Network Design* (Watson, Lewis, Cacioppi, Jayaraman, ISBN-13: 978-0133017373) http://networkdesignbook.com/
- The Wikipedia page on "embarrassingly parallel" problems is well refereed. http://en.wikipedia.org/wiki/Embarrassingly_parallel
- R. Solanki. Generating the noninferior set in mixed integer biobjective linear programs: an application to a location problem. *Computers and Operations Research*, 18:1–15, 1991.
- Ralphs, Ted, Saltzman, Matthew and Wiecek, Margaret. An improved algorithm for solving biobjective integer programs. *Annals of Operations Research*, 147(1):43-70, 2006.