AI Magazine Volume 29 Number 2 (2008) (© AAAI)

Lessons Learned Delivering Optimized Supply-Chain Planning to the Business World

James Crawford

■ *The late 1990s saw a huge upswing in the demand for supply-chain management software. The rapid commercial adoption of these tools brought into sharp relief the complexity of automating supply-chain planning and optimization. This article summarizes four years of real-world experience in combining AI and operations research (OR) techniques, balancing representational power against planning complexity, and understanding the practical implications of nondeterministic polynomial time (NP)-completeness.*

The late 1990s were a very interesting time for business enterprise software in general, and software for optimized supply-chain planning in particular. The spread of enterprise resource-planning software, warehouse-management systems, factory-planning systems, and other enterprise applications, had the side effect of moving online much of the raw data needed to optimize supply chains. Furthermore, the pace of business change, increased competitive pressures to react quickly to this change, and the rapid development of online commerce forced businesses to question the week-plus supply-chain planning cycles that had been the norm. Finally, the year 2000 (Y2K) problem caused an across-the-board replacement of enterprise software, allowing many businesses to update their approach to supply-chain planning.

The end result of all of these factors was a huge upswing in demand for supply-chain planning tools from i2 Technologies and other vendors. When I joined i2 in 1996 as optimization architect, the company had around 250 employees and roughly $100 million in annual revenues. At its peak in around 1999, i2 had grown to have a market capitalization of over $13 billion, had over 600 customers, and was listed on the Nasdaq 100.

This rate of growth brought into sharp relief two critical technical challenges in supply-chain management. First, the data required to manage supply chains was indeed online, but it was inevitably inaccurate and incomplete. Further, it resided in a host of incompatible and disconnected databases. Second, and equally important, supply-chain planning and optimization were a perpetual challenge.

There are solid theoretical reasons why supply-chain planning and optimization are hard. Technically the underlying optimization problem is either NP or P-space complete (depending on the details of the domain). Furthermore, the problem mixes a dozen or so classic optimization problems from AI and operations research (OR), and much of the expected savings from global supply-chain optimization are lost if these problems are treated independently.

This article describes our experience from four years of solving supply-chain planning and optimization problems across industries, and some of the lessons we learned.

Supply-Chain Modeling

In its broadest form, a supply chain consists of every activity performed by a manufacturing company from the time it purchases raw materials to the time it sells finished goods. This includes purchasing, storage, transportation, assembly or manufacture, testing, packaging, and delivery. The broad outlines of what makes a supply chain are common across many industries including semiconductor manufacture, wine production, furniture manufacture, steel production, oil refining, clothing, shoe manufacture, consumer packaged goods, and others.

During my time at i2, my team worked on problems from all of these industries. Further, some of our customers had reached an amazing degree of vertical integration across the extended supply chain. For example, one leading food manufacturer we worked with has a supply chain that starts with the production of special potato seeds, which are shipped to the farmers, and ends with vendor-managed inventory on the store shelves. Similarly, a large wine manufacturer we worked with has a supply chain that begins with the production of glass for the bottles and ends on the store shelf.

The language i2 developed to represent the constraints and optimization criterion in a supply chain was based on three primitive concepts: buffers, operations, and resources. "Buffers" capture the general notion of physical points in the supply chain at which a kind of good can be stored. This modeling object is used to represent bottles of wine in a wine cellar, screws in a bin by a machine, pallets of laptops on a warehouse floor, rods of steel, tanks of oil, and so on. A number of constraints are placed on buffers including size limitations, flow policies (for example, first in first out), safety stock policies (for example, that the buffer must always contain enough material to cover the next week of anticipated demand), and so on.

Certain buffers, generally at the end of the supply chain, hold finished goods—often goods that have been successfully delivered to customers' loading docks. At these buffers, optimization metrics are placed that measure, for example, lateness or shortness (lateness meaning that goods arrived into the finished goods buffer later than the date they had been promised to the customer and shortness meaning that an insufficient quantity of finished goods arrived).

The second primitive was the operation. "Operations" are any activity that turns one or more kinds of goods, taken from a set of buffers, into one or more kinds of goods, (which are again placed into a set of buffers). Operations could be anything from running a lathe on a factory floor, to assembling a computer, to loading pallets onto a truck, to a refining step in a chemical process, and so on. As with buffers, a number of constraints can be layered on top of operations, including lot size constraints, run times, start and stop times, efficiencies, and resource constraints.

"Resources" are the final basic primitive. Resources include any person or object needed to perform an operation. This could be anything from a particular machine, to a truck, to a special tool, to a trained operator, and so on. As with the other primitives, a large number of constraints could be placed on resources including capacity limits, availability calendars, and so on.

In some cases i2 developed very large supply-chain models. The general form of these models was a tree of operations and buffers fanning out from the finished goods up to the raw materials required for their production. Some of these trees could become 10, 20, or more levels deep and involve factories and warehouses spread across the world. Others could involve tens, or even hundreds of thousands of different finished goods, requiring interacting operations and resources for their production.

To i2's credit, i2's modeling language was rich and flexible enough to allow the company to model the flow of goods, the constraining resources, and the optimization metrics for supply chains across a very rich set of industries. Furthermore, unlike a linear or integer programming model, the models built using i2's language were directly accessible and intuitive to domain exerts and could be inspected graphically for accuracy and completeness. Finally, the models, together with the data imported into them to support supply-chain planning, provided a previously unknown level of visibility into the customers' supply chains—in many cases this visibility alone provided significant business value by allowing problems and bottlenecks in the supply chain to be much more quickly identified.

Supply-Chain Planning and Optimization

Unfortunately, there is a well-known trade-off between the power of a representation language and the complexity of constrained optimization of problems defined in the language (see for example, Helmert [2003]). i2's first successful commercial product was a factory planning product with restricted representational power and a set of heuristics-based optimization algorithms. The driving need that i2 addressed in designing its second-generation supply-chain planner was to create a system that could represent the much more complex constraints and optimization criterion in supply chains. However, insufficient thought was initially given to the problem of optimization over this new, more expressive language. This proved to be a significant challenge over the next several years.

Various reasons were given for thinking that supply-chain planning and optimization would not be "too difficult." First, it was argued that on large planning problems the human mind is as much at sea as an automated planner. Thus an automated planner need only reach a quality level better than a human planner. This sounded good in theory, but in practice we found that for many domains there were humans who had literally spent their lives learning how to optimize a particular part of the supply chain. While it may be true that they could not work on the entirety of the supply chain, they could certainly understand their part and were able to find flaws in the plans generated by our planner by proving that we were missing "obvious" optimizations.

A second, similar, argument was that global optimization, in the literal sense of creating a provably optimal plan, was not important. Many people argued that what was important was getting to a "pretty good" plan, and the last few percent of optimization were not worth the cost (in compute time or algorithmic complexity). This argument is almost right, and our successful approaches to supply-chain planning were approximate algorithms that generally did not yield provably optimal plans.

However, there are two considerations that any nonoptimal plan must address. First, as noted above, any plan must pass the test of not having flaws that are obvious to human experts. Since any plan that is not mathematically optimal will have flaws, this is sometimes a matter of trial and error. Further, real-world problems do have structure, and in many cases that structure seemed to conspire to cause our planner to generate materially suboptimal plans—that is, plans that, if not corrected, would cost our customers money.

To take just one example, many problems include a secondary, or often tertiary, optimization metric called "fairness." Intuitively *fairness* means that the plan distributes the "pain" of lateness or shortness "fairly" across customers instead of, for example, completely shorting one customer and making a full delivery to another. From a global perspective, fairness on one particular end item might seem to be a minor detail. However, if Wal-Mart and Target both order 100 pallets of PlayStations, but the plan ships 100 pallets to Wal-Mart and 50 to Target, then there may be major impacts on customer relations. A "fair" plan, by contrast would ship 75 pallets to each. This plan flaw may yield an extremely small difference in the global optimization metric, but could cause Target to cancel the contract. In other more complex cases, minor decisions upstream can interact with lot size, or timing constraints on operations, and yield major differences in the amount of material shipped to the end customers (or the timing of that shipment).

A third argument was that in the real world there is no such thing as planning—only replanning. That is, there is always a plan and the job of the "planning" system is to revise that plan in minor ways to account for some set of changes. In each such replanning event relatively little work will be needed because most of the plan will remain as it was. This line of thinking led i2 to initially focus heavily on repair-based approaches to planning.

Again this argument sounds great in theory. However in practice we ran into a number of problems. First, the first thing each customer wanted to see was for our system to generate a plan for meeting some set of customer orders. They seemed unconvinced that this was not going to be the normal mode of interacting with the system. Thus we were inevitably forced to confront the "from scratch" planning problem in nearly every account. Second, replanning for supply chains turns out to be every bit as hard as planning. The details are beyond the scope of this article, but the technical core of the problem is that the supply-chain planning problem, under most obvious hill-climbing metrics, has extremely large plateaus over which the metric does not change at all. These plateaus make it challenging to utilize hill-climbing or other repair methods. Intuitively these plateaus arise because optimization metrics are generally measured at the end of the supply chain (on the finished goods buffers) but most of the interesting decisions are made upstream when allocating time on critical resources in any of several factories.

Finally, the size of problems that we needed to address took all of us by surprise. At one point my research team developed a set of data structures that allowed us to perform extremely fast plan generation. We hoped that these data structures would allow us to execute thousands or, ideally, hundreds of thousands, of repair actions on our plans. Instead our sales force took our prototype and used it to solve problems involving literally millions of separate activities on worldwide supply chains. The good news was that we were able to solve problems larger than almost all of those attacked previously using optimization technology. The bad news was that the problems were so large that we could not run the plan builder enough times to get any meaningful optimization through local search (this was partially due to the fact that we were able to create very good heuristics to guide the construction of the initial plan).

As a result of these challenges, in around 1997 our teams at i2 were looking for creative solutions for some of the supply-chain planning problems we faced. The approach that worked best, unsurprisingly, was to develop highly procedural algorithms on a customer-specific basis. This was done using a combination of heuristics and domain-spe-

Often, It's Not about the AI

Neil Jacobstein

Narrowly focused task- and domain-specific AI has been applied successfully for more than 25 years and has produced immense value in industry and government. It doesn't lead directly to artificial general intelligence (AGI), but it does have real problem-solving value. It is useful to note that many of the reasons some otherwise meritorious AI applications fail have nothing to do with the AI per se but rather with systems engineering and organizational issues. For example, the domain expert is pulled out to work on more critical projects; the application champion rotates out of his or her position; or the sponsor changes priorities. A system may not make it past an initial pilot test for logistical rather than substantive technical reasons. Some embedded AI systems may work well for years on a software platform that is orphaned, and porting it would be prohibitively expensive. A system may work well in a pilot test, but it might not scale for huge numbers of users without extensive performance optimization. The core AI system may be fine, but the user interface could be suboptimal for a new set of users. The delivered application system might work well, but it could be hard to maintain internally. The system may work according to the sponsor's requirements, but it might not be applied to the part of the problem that delivers the largest economic results; or the system might not produce enough visible organizational benefits to protect it in subsequent budget battles. Alternatively, the documented results may be quite strong, but might not be communicated effectively across organizational boundaries. All software projects, with or without embedded AI methods, are vulnerable to one or more of these problems.

Neil Jacobstein is president and chief executive officer of Teknowledge Corporation in Palo Alto, California, and a visiting scholar in the MediaX Program at Stanford.

cific algorithms (many drawn from operations research). In addition, i2 began to use linear programing to solve a number of subproblems in supply-chain planning—for example in logistics and for certain steps in the supply chain (such as down-binning in semiconductor manufacture).

While these methods were largely successful, they were not obviously scalable due to the custom work required on each account.

Fortunately, we fairly quickly began to recognize broad patterns of which kinds of algorithms were applicable to which supply chains. This led us to develop templates for each industry. This started with computer manufacture (at this point i2 had huge presence in the computer manufacture segment) and semiconductor manufacture, but eventually spread to apparel, steel, consumer packaged goods, and other industries. The development of industry templates was an extremely important step for i2. It not only helped us address the optimization problem, but also greatly simplified the problem of modeling each customer's supply chain (since the template included a model of a typical supply chain for each industry) and encapsulated best-practices in supply-chain planning for each industry (which was a major source of value to our customers). Finally, we were able over time to rearchitect the core of the software to make it relatively easy for industry-specific development teams to develop and support the optimization algorithms in each template.

Interesting, these templates often mixed procedural planning algorithms, linear programming, integer programming, and plan repair. They also generally involved running a series of algorithms on different parts of the supply chain. Finally, the hardest problems, in domains such as chemical manufacture, often involved a deep interaction between liner programming and discrete optimization (for example discrete decisions around lot sizing and campaign planning coupled with linear optimization for each step in the campaign). Nevertheless, in each industry we were able to, over time, develop largely successful planning and optimization approaches.

Lessons Learned

In retrospect we can see several interesting lessons.

First, many, if not most, of the optimization problems we encountered eventually required a combination of AI and OR techniques. This may not be a major surprise today since the two communities have been working increasingly closely over time (see for example, Kautz and Walser [2000]; Van den Briel and Kambhmpati [2005]; Van den Briel, Vossen, and Kambhmpati [2005]; and Vossen et al. [2001]), but it was an important lesson at the time.

Second, it turned out to be important to separate the data structures for modeling from the data structures for optimization algorithms. The overall supply-chain planning problem must be represented in a way that is holistic and natural for the user (and that roughly matches the ontology of the data coming in from other systems and going out to other systems). However, we found that both AI repair algorithms and linear programming solvers could be efficient only if they could use their own ontology and data structures. This forces a certain amount of translation back and forth (especially when the solvers must work together), but all of our attempts to avoid this cost were unsuccessful.

Third, as mentioned above, generic modeling languages inevitably mean complex optimization. Limited languages, such as linear programming, support provably optimal polynomial time solvers. Other restricted problems, such as "traveling salesman" or "job-shop scheduling," have been well studied and can be solved near-optimally up to fairly large problems. However, supply chains are complex, and any attempt to restrict the modeling language led to cases where we could not model the customer's situation with sufficient precision (leading to hugely suboptimal or infeasible plans).

In a real sense, the power of the industry templates was that they limited modeling flexibility at a semantic level. They fixed, for example, a basic structure of buffers, operations, and resources that was common across semiconductor manufacture (and the general constraints on each object). This allowed each template to have its own heuristic optimization algorithms that would not work on all supply chains but would work on supply chains having the structure given in the template.

Finally, our overall experience gives a new perspective on the meaning of NP-completeness. We came to appreciate the use of linear programming not because it was necessarily faster or more optimal than AI methods, but rather because it was more predictable. When linear programming worked on small test problems it generally also worked on larger production problems (no surprise there). Further, when it worked on one customer, or for one time period, it generally also worked on others. Search-based, or repair-based, techniques, by contrast, had to be watched much more closely. When the problem size changed, the data set changed, or the field consultants made innocuous changes, algorithms that had been working would suddenly either run unacceptably long or would generate obviously flawed plans (which again makes sense given the theoretical results). In the business world, the ability to scale solutions, to test and deploy solutions that we know to work, and not to require Ph.D.-level assistance in every account is extremely important. Otherwise costs and schedules cannot be accurately estimated, and it is difficult to run a profitable operation. These considerations turn out to be much more important than the generation of provably optimal solutions.

Conclusion

The late 1990s saw an interesting collision between the needs of the business world to solve hard optimization problems, such as supply-chain planning, and the computational complexity of these problems. In the end, developing general-purpose, fully dependable solutions that worked across all industries proved an elusive goal. A number of lessons were learned including the practical implications of NP-completeness.

In response to these lessons, a set of industry-specific algorithms was developed and deployed broadly with good results. i2, in particular, was arguably the most successful company in the industry in providing optimized, though not provably optimal, plans across a very rich set of industries. These algorithms combined heuristics with OR techniques and AI techniques. The end result was a measurable increase in supply-chain efficiency including decreased inventory and increased flexibility and resource utilization.

References

Helmert, M. 2003. Complexity Results for Standard Benchmark Domains in Planning. *Artificial Intelligence* 143(2): 219–262.

Kautz, H., and Walser, J. P. 2000. Integer Optimization Models of AI Planning Problems. *Knowledge Engineering Review* 15(1): 101–117 (www.cs.rochester.edu/u/kautz/papers/kerilp.ps).

Van den Briel, M., and Kambhmpati, S. 2005. Optiplan: A Planning System That Unifies Integer Programming with Planning Graph. *Journal of Artificial Intelligence Research* 24: 919–931.

Van den Briel, M.; Vossen, T.; and Kambhmpati, S. 2005. Reviving Integer Programming Approaches for AI Planning: A Branch-and-Cut Framework. In *Proceedings of the International Conference on Planning and Scheduling,* 310–319. Menlo Park, CA: Association for the Advancement of Artificial Intelligence.

Vossen, T.; Ball, M. O.; Lotem, A.; and Nau D. S. 2001. Applying Integer Programming to AI Planning. *Knowledge Engineering Review* 16(1): 85–100.

James Crawford is vice president of engineering at Composite Software, an enterprise software startup in the San Francisco Bay Area. Prior to working at Composite, Crawford spent three years as the area lead for autonomy and robotics at NASA Ames Research Center. Among many other projects one of his teams delivered the optimized activity planner used throughout both of the Mars rover missions. Prior to joining NASA, Crawford was optimization architect for the Supply Chain Planner at i2 Technologies and led i2's optimization team. Before that, he worked at AT&T Bell Laboratories and cofounded the Computational Intelligence Research Laboratory (CIRL) at the University of Oregon. Crawford has authored over 15 papers in referred journals and conferences, and holds five patents. He holds a Ph.D. and master's degree in computer science from the University of Texas at Austin and a B.A. in mathematics and computer science from Rice University.