Preprint
UCRL-JC-135996

Approximating the 0-1 Multiple Knapsack Problem with Agent Decomposition and Market Negotiation

B.A. Smolinski

This article was submitted to
13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems
New Orleans, LA
June 19-22, 2000

September 3, 1999

U.S. Department of Energy
Lawrence Livermore National Laboratory

Approximating the 0-1 Multiple Knapsack Problem with Agent Decomposition and Market Negotiation

Brent A. Smolinski
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
7000 East Ave, Livermore, CA 94550-9234 USA
smolinski1@llnl.gov

Abstract

The 0-1 multiple knapsack problem appears in many domains from financial portfolio management to cargo ship stowing. Methods for solving it range from approximate algorithms, such as greedy algorithms, to exact algorithms, such as branch and bound. Approximate algorithms have no bounds on how poorly they perform and exact algorithms can suffer from exponential time and space complexities with large data sets. This paper introduces a market model based on agent decomposition and market auctions for approximating the 0-1 multiple knapsack problem, and an algorithm that implements the model (M(x)). M(x) traverses the solution space rather than getting caught in a local maximum, overcoming an inherent problem of many greedy algorithms. The use of agents ensures that infeasible solutions are not considered while traversing the solution space and that traversal of the solution space is not just random, but is also directed. M(x) is compared to a bound and bound algorithm (BB) and a simple greedy algorithm with a random shuffle (G(x)). The results suggest that M(x) is a good algorithm for approximating the 0-1 Multiple Knapsack problem. M(x) almost always found solutions that were close to optimal in a fraction of the time it took BB to run and with much less memory on large test data sets. M(x) usually performed better than G(x) on hard problems with correlated data.

Keywords: Market Algorithm, Autonomous Agents, Agent Decomposition, Market Negotiation, Knapsack Problem, Heuristic.

1. Introduction

The 0-1 multiple knapsack problem is a generalization of the 0-1 Knapsack problem arising when *m* containers (or knapsacks), of given capacities c_i (i = 1, … , m) are available, along with *n* items, of which each item has a value v_j and size $s_j > 0$ (j = 1, … , n). These *m* knapsacks are to be filled with some or all of the *n* items such that each item is placed in at most one knapsack, all items placed in a particular knapsack fit in that knapsack, and the total value of all items placed in all of the knapsacks is maximized. A precise definition of the 0-1 multiple-knapsack, as stated in (Martello and Toth 1990), is:

$$\text{maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} v_j x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} s_j x_{ij} \le c_i, \quad i \in M = \{1, \ldots, m\}$$

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad j \in N = \{1, \ldots, n\}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i \in M,\ j \in N$$

where x_{ij} = 1 if item j is assigned to knapsack i; 0 otherwise. The 0-1 multiple knapsack problem is NP-hard in the strong sense.

Multiple knapsack problems appear in such domains as financial portfolio management and naval ship stowing. For example, the CAD Research Center[1] has developed a software program, ICODES (Pohl et al. 1997), to aid Naval and Marine stow planners with stowing cargo items onto ships. A ship has *m* stow areas (knapsacks) each with capacity c_i $i \in \{1, \ldots, m\}$. A cargo list has *n* items each with a value v_j and size s_j $j \in \{1, \ldots, n\}$, where v_j is determined by loading priorities (items with a higher loading priority will have a higher preciousness $p_j = v_j / s_j$). The goal for stowing a ship is to maximize the total value, determined by the loading priorities, onto the ship, while satisfying a number of constraints: items fit, an item can be stowed in at most one stow area, trim, stability and weight allocation constraints are satisfied, etc. Typical problems in this domain have thousands of cargo items and dozens of stow areas.

This research develops a model for approximating the 0-1 multiple knapsack problem based on agent decomposition and market negotiation. A simple algorithm that implements this model, M(x), is compared to a bound and bound algorithm (BB) and a greedy algorithm with a random shuffle (G(x)). The results suggest that M(x) is a good algorithm for approximating the 0-1 Multiple

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48. This work has been funded by LDRD UCRL-JC135996.

[1] CADRC, Cal Poly, San Luis Obispo, CA 93407; www.cadrc.calpoly.edu

Knapsack problem. It almost always found solutions that were close to optimal in a fraction of the time it took BB to run and with much less memory on large test data sets. M(x) usually performed better than G(x), especially on hard problems with correlated data ($p_j = 1$ for $j \in \{1,\ldots,n\}$). Hard problems are where the number of rounds for G(x) to converge to a desirable solution is greater than a few rounds. M(x) was able to converge to good solutions with fewer rounds than G(x).

This paper is organized as follows. Section 2 presents approaches to solving 0-1 multiple knapsack problems, and highlights problems with them. Section 3 discusses the use of markets to negotiate for resources. It suggests that a method based on agent decomposition may work well for approximating the 0-1 multiple knapsack problem if a market paradigm is used. Section 4 describes a market-oriented model and algorithm for approximating the 0-1 multiple knapsack problem based on agent decomposition and auction negotiation mechanisms. The model is a computational market based on a sealed bid and a continuous auction (Steiglitz et al. 1996 and Engelbrecht-Wiggans et al. 1983). Section 5 describes the results obtained from running M(x) and other algorithms on a suite of test data. Conclusions and future work are given in Section 6.

2 Methods

The simplest approach used to solve the 0-1 multiple knapsack problem is a greedy first fit: an item is placed into the first knapsack that can hold it (Garey and Johnson 1979 and Martello and Toth 1990). This algorithm runs relatively fast, however, rarely achieves an optimal allocation and has no lower bound on performance.

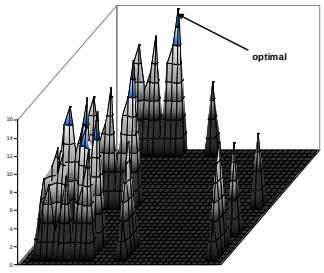
To find better solutions to this problem, other approaches are used. Two deterministic approaches used to solve the 0-1 multiple knapsack problem are branch and bound (Markland 1989) and bound and bound (Martello and Toth 1985). Both of these algorithms are guaranteed to find an optimal solution. However, they both have time complexities with upper bounds that are $O(m^n)$. Also, assuming a surrogate relaxation technique is used to calculate the upper bound at the decision nodes, they both have space complexities that are $O(nC)$, where C is the size of the knapsack solved with the surrogate relaxation problem.

What is desired is a method that will produce solutions that are better than those found by simple greedy algorithms, yet with time and space complexities which are not exponential in the size of the problem. Choices include a hill climbing algorithm and a stochastic search algorithm (e.g. genetic algorithm). Unfortunately, as Figure 1 suggests, multiple knapsack problems tend to have jagged, discontinuous search surfaces with isolated global optimums. Hill climbing algorithms work poorly on search surfaces such as these where derivatives do not always exist. Stochastic methods, such as genetic algorithms, tend to have a very difficult time finding optimal solutions on sparse surfaces where the global optimum is isolated. It is usually by sheer luck that these algorithms find the optimal solution under these conditions (Goldberg 1989 and Michalewicz 1992).

One approach to solving complex problems is to divide a single problem into several, smaller problems, then solve each sub-problem independently. The result is obtained by combining all of the individual solutions. Using this approach each knapsack could be considered individually to find the optimal solution for each, thus reducing the computational complexity and memory requirements. However, because individual solutions may be inter-dependent, this will not always provide a valid, much less an optimal, answer. For example, an item could be in the solution set for more than one knapsack. Constraints for the whole problem must be respected during problem decomposition.

To satisfy inter-dependent constraints among the sub-problems, they must be solved together. Objects in the problem domain can be modeled as autonomous agents, or object- agents (Aly 1994, Eastman et al. 1992, and Pohl et al. 1997) where the interests (or goals) of the agents are consistent with the goals and constraints of the whole problem. For the 0-1 multiple knapsack problem, each knapsack will have an agent assigned with interests in acquiring certain items. Each agent tries to maximize the value of the items selected, subject to the constraint that they fit in their knapsack. There are well known pseudo-polynomial time exact algorithms and polynomial time approximation algorithms to solve individual 0-1 knapsack problems. However, similar to the divide and conquer method, finding a feasible solution for each agent does not guarantee a feasible solution to the whole problem. It is possible that more than one agent will want the same item(s) in its solution set.

When conflicts arise, agents must negotiate with each other to resolve them. While much research has been conducted on conflict resolution (Davis and Smith 1983, Durfee 1988, Durfee and Lesser 1987, 1989, Roth 1985, Zlotkin and Rosenschein 1996), many of these approaches



Two knapsacks c_i where c_i of size knapsack i:
$M = \{5, 4\}$

10 items (s_j, v_j) where s_j = size of item j and v_j = value of item j:
$N = \{(2,2), (2,6), (3,3), (3,6), (4,4), (4,4), (5,5), (5,5)\}$

FIGURE 1. PARTIAL SOLUTION SPACE FOR A SAMPLE 0-1 MULTIPLE KNAPSACK PROBLEM

rely upon negotiation through the exchange of partial plans, which may generally be very difficult to develop for this problem. Chapman (1987) showed that general planning, based on his TWEAK representation, is undecidable even with finite initial states. Remembering that the reason for decomposing the problem was to produce a method for solving the problem that is fast and effective, this negotiation method may be ineffective because it is not guaranteed to be fast (or even to halt). For a method of a agent decomposition to be effective, a fast, possibly approximate, negotiation mechanism is needed.

3. Using Markets for Negotiation

In the previous section, it was shown that decomposing the problem left us with a reconstruction problem that is just as hard to solve as the original problem. For a "divide-and-conquer" method to be successful, the sub-problems must be naturally disjoint. The sub-problems that arise with this method of problem decomposition are not disjoint. It is believed that the complexity of problems in the classes NP-Hard are due to the inability to naturally decompose the problems further. However, agent decomposition can still be used by using a market heuristic for negotiation. The nice thing about markets is that they can achieve their effects through simple interactions. They are able to solve complex resource allocation problems with very little information (prices), which makes them a desirable approach for resolving conflicts that arise when allocating resources (Clearwater 1996)

Markets may not always provide the best means for allocation. For example, the existence of firms suggests that centrally planned problem solving paradigms may work better on smaller resource allocation problems. Intuitively, this makes sense. Even though a problem may theoretically be computationally complex (i.e. NP-Hard), small instances of the problem may be solved in a reasonable amount of time. This paper demonstrates that the market paradigm works well for multiple knapsack problems whose size is sufficiently large. This is because the running time for M(x) is $O(mnN)$ (N = number of rounds), where as BB exhibits much greater average running times for larger problems (BB has a worst case running time $O(m^n)$). M(x) is not guaranteed to find the globally optimal solution, but compared to BB, it performs well on large test data sets.

3.1 Market Applications

Market-oriented algorithms have been applied successfully towards solving complex resource allocation problems such as distributed multi-commodity flow in the trucking industry (Wellman 1993, 1996), query processing and data migration in a distributed database (Stonebraker et al. 1994), task scheduling in a distributed operating system (Huberman 1995, Malone et al. 1988, Miller and Drexler 1988), network and file system resource allocation (Ferguson et al. 1988, Gagliano and Mitchem 1996, Kurose and Simha 1989, Kuwabara et al. 1996, Miller et al. 1996, and Yemini 1981), allocating tradeable pollution permits (Marron and Bartels 1996), task allocation in discrete manufacturing systems (Baker 1996, Tilley 1996), and load balancing in distributed systems (Ferguson et al. 1996). Most of the current research has focused on developing one of two types of economic models. One is an exchange based economy (Sandholm 1993), the other is a price based economy (Miller and Drexler 1988, Stonebraker et al. 1994, Waldspurger et al. 1992, and Wellman 1993, 1996). Neither model fits the 0-1 multiple knapsack problem, where an approach using auctions is needed (Miller et al. 1996 and Steiglitz et al. 1996).

3.2 Market Types

Most market oriented approaches used to solve resource allocation problems use one of three types of models: price based economy, exchange based economy, and an auction.

In a price based economy producers and consumers participate in a market for goods and resources. Producers buy resources and transform them into commodities and sell them to consumers. Producers may also be consumers. Goods and resources are allocated through a market place in which equilibrium is determined by pricing mechanisms. Algorithms of this genre are typically used to solve problems of scheduling and resource allocation over time, where the elements of the problem are continuously changing and naturally distributed. For example, in Wellman (1993) this model is used to solve a multicommodity flow problem where the resources are geographically distributed and constantly changing. These algorithms are not appropriate for solving the 0-1 multiple knapsack problem since all items in the problem domain are known and constant. Thus, the modeling of producers does not make sense.

In an exchange based approach each agent is endowed with limited resources. The agents exchange these resources until their marginal rates of substitution are equal. They only exchange resources in the direction of increasing overall utility. In other words, they exchange resources such that at least one agent is made better off and no agent is made worse off. A Pareto optimal allocation is achieved when no exchange can take place without making some agent worse off. The problem with this approach is that it must converge to an equilibrium point, which may be far from a globally optimal solution. The market is unable to move from this equilibrium point because exchanges are constrained to occur in the direction of increasing utility. The rate of convergence degrades catastrophically (with discrete optimization problems) as the number of items

increases because in the worst case, all combinations of exchanges will have to be considered before equilibrium is reached.

The auction market is similar to a price based economy but withoug producers. Agents bid for the items offered in a central market. Prices are determined through *tatonnement* or *non-tatonnement* processes. In *tatonnement* processes (Walras 1954 and Cheng and Wellman 1998), each agent is endowed with specific initial wealth. Initial prices for resources are set to arbitrarily. An agent computes its demand for a resource based on its utility function and budget constraints. It sends its demand to a central auctioneer, which in turn computes the aggregate demand for the resources. If excess demand is positive, the price of the resource is incremented. If excess demand is negative, the price is lowered. Once the price is adjusted, excess demand is computed again. This process continues until supply exactly equals demand (equilibrium is reached). Determining an equilibrium price can be computationally expensive because an unknown number of price postings and aggregate demand calculations must take place before equilibrium is reached. In *non-tatonnement* processes, agents are allowed to trade before the economy (or market) has reached equilibrium. Such a process is fast because trading begins before equilibrium is reached. A possible disadvantage is that intermediate trading never decreases an agent's utility. As in an exchange based economy, this can result in a Pareto equilibrium allocation that is not a global optimum (a local optimum of the solution surface).

Auction strategies include single sided call auction, sealed bid auction, continuous auction, and double auction (Scarf 1984 and Steiglitz et al. 1996). The method described in this paper is a combination of a sealed bid and continuous auction. Algorithms using sealed bid auctions to solve the 0-1 multiple knapsack are simple and fast because a non-tatonnement process of calculating a clearing price is used. Highest bids are selling. A variation of a continuous auction approach is integrated into the algorithm. Agents may trade and sell items back to the auctioneer in hope of getting a better allocation of resources. By relaxing the constraint that every trade results in no agent being made worse off, the algorithm may traverse the entire solution space.

4. Developing a Market Oriented Algorithm

M(x) implements a model, based on agent decomposition and market negotiation. That model includes consumers, an auctioneer, items, and mechanisms for trade amongst the agents and the auctioneer. M(x) implements the auctioneer's and consumers' behavior as well as protocols for the trading mechanisms within this model.

The computational market in M(x) uses both a sealed bid auction and a continuous auction (Marron and Bartels 1996). In a sealed bid auction the auctioneer first collects bids for groups of items from all the consumers, then determines the clearing price. Consumers who bid at the clearing price or higher get to purchase the items. M(x) implements a slight change where items are considered one at a time. This makes for a simple calculation of the clearing price: the highest bid. In a continuous auction, agents continuously post bid and ask prices for items. M(x) has a phase similar to a continuous auction when items are either sold back to the market or are swapped between consumers. Without this phase M(x) would be a variation of a greedy algorithm. Because the auction is continuous, it is able to traverse the entire solution space. A greedy algorithm might stop on a local maximum.

Individual agents try to maximize utility by bidding for items that have high utility for them. An item's value is the same regardless in which knapsack it is placed, but utility for an item is based on preferences, which vary over time and between agents. The bidding strategy implemented by M(x) assumes consumers prefer small precious items. A consumer would be willing to bid higher for such items. Items that do not fit into a knapsack have no utility for the associated consumer. The auctioneer tries to maximize the total sales for items. This research describes one bidding strategy and one market mechanism, though other combinations of bidding strategies and market mechanisms can be implemented using the market model developed in this research.

4.2 Market Model

The model is shown in Figure 2. It describes an auction and consumers. The auction consists of an agent (the auctioneer) and the items to be auctioned. Each consumer represents a knapsack. Interactions between agents are one of three types: 1) consumers purchase items from the

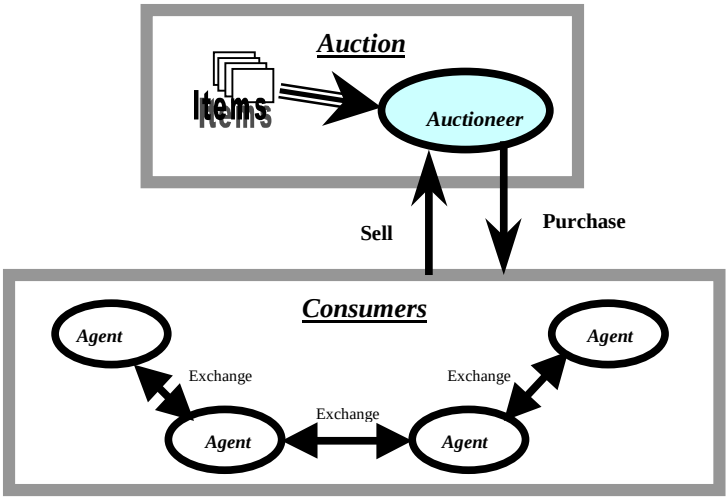


FIGURE 2. MARKET MODEL.

auctioneer (Purchase protocol), 2) consumers sell items to the auctioneer (Sell protocol), 3) consumers swap items with each other (Exchange protocol). The Purchase protocol defines how items will be allocated through the auction. The Sell and Exchange protocols allow algorithms that implement the model to traverse the solution space. Each algorithm is free to define the particular mechanics of agent behavior and exchange.

4.3 Purchase Protocol

This section describes the purchase protocol in M(x). It is assumed that a consumer will prefer items with larger preciousness and smaller size. Items that do not fit into the knapsack will not be bid upon. The utility consumer *i* receives from having item *j* (U_{ij}), after item *j* has been placed in its knapsack is:

$$U_{ij} = {p_j(a_i)}/{c_i} \quad \text{where}$$

$p_j = v_j / s_j$ (p_j is the preciousness of item *j*) and

$b_i = \sum_{j=1}^{n} s_j x_{ij}$ (sum of item sizes in consumer *i*'s knapsack)

and

$a_i = c_i - b_i$

The utility consumer *i* will receive from consuming item *j* (U'_{ij}), before item *j* has been purchased is:

$$U'_{ij} = \begin{cases} 0 & \text{if } b_i + s_j > c_i \\ \left(p_j \left(\frac{a_i + s_j}{c_i} \right) + 1 \right) & \text{if } b_i + s_j \le c_i \end{cases}$$

Assuming that consumer *i* has an initial wealth, or endowment of resources r_i:

$r_i = \eta_i + \rho_i \left(\frac{\eta_i(\eta_i - 1)}{2\eta_i} \right)$, for $i = 1 \ldots m$, where

$\rho_i = max(p_j (min(\lfloor c_i / s_j \rfloor, 1))$ for $j = (1 \ldots n)$ and

$\eta_i = \sum_{j=1}^{n} (min(\lfloor c_i / s_j \rfloor, 1))$

and B_{ij}, the bid consumer *i* makes for item *j*, equals U'_{ij}, then the total amount consumer *i* spends can be bound by:

$$0 \le \sum_{j=1}^{n} B_{ij} \le \eta_i + \rho_i \left(\frac{\eta_i(\eta_i - 1)}{2\eta_i} \right)$$

This means every consumer has enough resources to bid for all the items that form the optimal solution and their bidding strategy is consistent with their initial endowment of resources r_i. This bidding strategy is simple, does not require agents to build internal models of the state of the system, and makes bid calculations fast.

The auctioneer simply tries to maximize its profit *P*:

$$P = \sum_{i=1}^{m} \sum_{j=1}^{n} B_{ij} x_{ij}$$

This is achieved by selling the most precious items first. The auctioneer sorts the items in decreasing preciousness prior to bidding, allowing convergence to a good solution quicker when the size of the problem is large. Of course, this introduces a bias towards placing items with higher preciousness into the solution space first, which may cause the algorithm to converge to a sub-optimal solution.

As an example, consider the configuration in Figure 3. If the next item to be bid on has size 5 and value 10, consumer 1 would bid 0 since the item would not fit into its knapsack, and consumer 2 would bid $\left(\frac{0}{5}\right)\left(\frac{5+5}{10}\right) + 1 = 1$ for the item. Thus the item is sold for 1 unit to consumer 2.

With the agents' behavior defined, a purchasing mechanism needs to be developed. The Purchase protocol is based on a sealed bid auction. The clearing price for an item is determined by the highest bidder, where items enter a market one at a time. If there are no bids greater than zero, the item is not sold. This process continues until all items have entered the market exactly once.

4.3 Sell Protocol

Using only the purchase protocol can find good results with some data sets, however, it is more likely to find a local minimum. It would be desirable for the algorithm to traverse the solution space. To do this, a sell protocol is introduced where after every round of bidding, each consumer is allowed to sell back some of its items to the market (POST procedure). After the items are sold back, they are placed at the end of *items* (the complete list of items). The agents then enter into another round of bids,

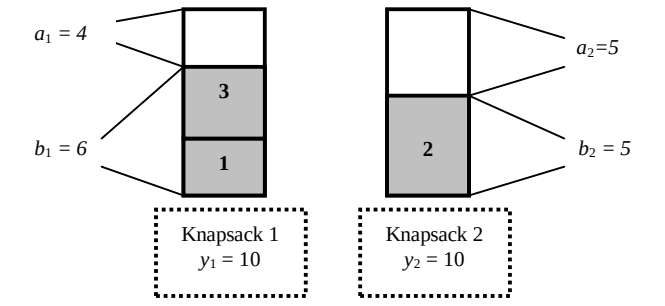


FIGURE 3. EXAMPLE CONFIGURATION.

and so on for *numrounds*. It is assumed, for now, consumers will choose to sell back their least precious items first. If the number of items sold back each round is constant, it is likely that the algorithm will get stuck in a local neighborhood, which suggests that maybe the number of items sold back should vary from round to round. The number of items sold back to the market in M(x) decreases with subsequent rounds. This is similar in principle to a simulated annealing algorithm where configurations stabilize over time (Kirkpatrick et al. 1983, van Laarhover and Aarts 1987, Varanelli and Cohoon1993). M(x) will make larger surface jumps in the beginning rounds and smaller jumps in the later rounds.

Because the goal was to develop a fast algorithm, M(x) implements a method for calculating the price an item is sold back to the auctioneer that requires little computation. Assuming that the auctioneer buys an item back for an amount equal to what it was previously sold, and assuming the auctioneer will always buy back items posted by a consumer, the algorithm does not have to compute the sell back price and keep track of individual budgets. This is because of the assumption that each consumer has enough initial resources to purchase its optimal solution and that its bidding strategy is consistent with and will never exceed its budget. If bid and sell back prices varied, then consumers would have to alter their bidding strategies as their budgets fluctuate. This would introduce a layer of complexity that would ultimately translate into computational complexity.

4.3.1 Random Behavior in Sell Protocol

Random behavior is not desirable under unbounded rationality. However, an agent's knowledge of the solution space is limited because unbounded rationality would dictate unbounded computational resources. Under bounded rationality an algorithm can get stuck in a local neighborhood in the solution space. One way to move from this neighborhood would be to introduce random behavior, which results in occasional random walks across the solution space.

M(x) incorporates this behavior. Two things happen every *sell_random* rounds in the POST procedure. First, the number of items thrown out every round (*numthrows*: such that *kround % sell_random* $\neq 0$, where *kround* is the current round) is decremented by *numthrowdec*. Second, *numthrows*/2 items are sold back by every consumer (choosing *numthrows*/2 items prevented all items from all knapsacks from being sold back on any particular round *kround*: *kround % sell_random = 0*). The variables *sell_random* and *numthrows* are initialized before bidding begins. The *sell_random* parameter should have a value that is about twice the logarithm of the number of rounds with an upper bound around 30 and a lower bound around 9 (which produced the best results for the particular data this algorithm was run on). Also, a value for *numthrows* should be chosen such that the number of items thrown out the first round from any knapsack will be on average between 50-100% of the items in the knapsack. With larger ratios of *n / m*, the average number of items thrown out in the first rounds should be close to 100%. This puts all items into the solution space quickly. With smaller ratios of *n / m*, the average number of items thrown out in the first rounds should be close to 50%. This mixes up the ordering of the items in *items* quickly. It is important to give all items (as well as all combination of items) an opportunity to enter the solution space. The number of items thrown out the last round should be close to one, but not less than one. This means a value of *numthrowdec* is wanted:

$numthrows - numthrowdec(\ numrounds\ /\ sell\_random\ ) \geq 1.$

Solving for *numthrowdec*:

$(\ numthrows - 1\ )\ /\ (\ numrounds\ /\ sell\_random\ ) \geq numthrowdec$

$\Rightarrow numthrowdec = \lceil numthrows\ /(numrounds\ /\ sell\_random) \rceil - 1.$

4.4 Exchange Protocol

Not all transactions in an economy are done through the market place. For instance, neighbors often exchange things like tools, sugar, salt. These transactions do not require pricing mechanisms, but simply take place through direct trade. Without these trades, convergence to an optimal solution can take very long. For instance, much time and effort is saved by borrowing sugar from a neighbor vs. making a special trip to the store to buy it.

In the POST procedure of M(x), every *exchange* rounds consumers swaps *numthrows* random items with their nearest neighbor, defined by the neighbors directly before and directly after them in the list of agents. The value of *exchange* should be chosen such that $exchange \neq sell\_random$ and $1 \leq exchange \leq 4$ (M(x) performed best with this value of *exchange*). One possible reason for such a small value of *exchange* is that frequent bartering gives the opportunity for all items to be placed in all knapsacks.

4.4 Final Algorithm

The final algorithm is shown in Listing 1. The time complexity of M(x) is $O(numrounds \times n \times m)$ (or $O(n \log n)$ for sort, whichever is larger) and the space complexity is $O(n \times m)$. However, the space complexity can be reduced to $O(n + m)$ if the state is represented by a single dimensional array with *n* structures, where each structure represents one item and has a field that signifies which knapsack that item is stored in. M(x) stores the state in a $n \times m$ matrix.

5 Experimental Results

This section analyzes the results from running M(x) over a complete test suite. M(x) is compared to two algorithms: A bound and bound algorithm, BB, (Martello and Toth 1985), and a simple first fit greedy algorithm with a random shuffle, G(x).

The test data generation algorithm used is described in Martello and Toth (1990). Tests were run on 100 different types of data, with 20 different data sets for each type. The types were broken up into two different groups of data with 50 different *n/m* ratios. The two groups were strongly correlated and uncorrelated. The strongly correlated data types had values of $s_I = v_I$ which was uniformly random in [1, 100]. The uncorrelated data types had values of s_I and v_I which were both uniformly random in [1, 100]. Within these three groups there were 50 different types comprised of all combinations of N = {20, 40, 60, 80, 100, 120, 140, 160, 180, 200} and M = {2, 4, 6, 8, 10}. The capacities of the knapsacks were calculated as:

$$c_i \text{ uniformly random in } \left[0, \left(0.5\sum_{j=1}^{n} sj \quad -\sum_{k=1}^{i-1} ck\right)\right]$$

for i =1, …, m – 1, with the capacity of the mth knapsack set to:

$$c_m = \left(0.5\sum_{j=1}^{n} sj \quad -\sum_{k=1}^{m-1} ck\right)$$

The size of the test data was limited to N = 200 and M = 10. The reason for this is that the calculation of the upper bounds in BB required $O(nC)$ space, where C is the size of the knapsack in the surrogate relaxation problem. Since C grows in size as n grows (from test generation algorithm), the size of the problem that could be solved by BB was limited by the development environment.

From Figure 4, it is clear that the relative running time of M(x) vs. BB decreases as the number of items increases. The same thing does not appear to happen with respect to an increase in the number of knapsacks. It looks as if the relative running times grows polynomially (and may even decline after $m \approx [8,10]$) as m increases. However, one can claim that as the problem size grows, M(x)'s relative running times to BB decline.

BB works very well on smaller data sets. When the problem size is small, BB performed better than M(x), where M(x) had average running times as much as 33 times BB, to get within 1% of error. This is due in part to the data itself, where a solution within 1% of optimal meant the optimal solution. Therefore, M(x) had to run until the optimal was found. On larger test data M(x) was a clear

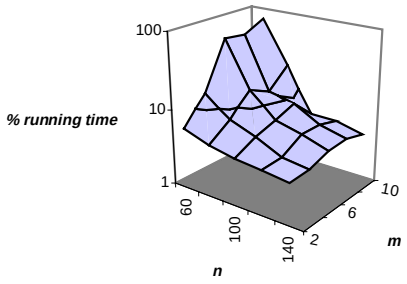


FIGURE 4. AVERAGE PERCENT RUNNING TIME OF M(X) VS. BB(-1) ON CORRELATED DATA OVER 20 DATA SETS TO GET WITHIN 1% ERROR (LOG SCALE).

winner, where average running times for M(x) were a fraction of BB, even on tighter error constraints.

Experiments were also run on M(250) and BB using a similar test suite. In most cases M(250) performed well on correlated data. The mean result was usually within 1% of the optimal solution (in many cases the median percent error was 0%) and the running times for M(250) were a fraction of BB on the larger data sets.

There was a noticeable difference in performance between correlated and uncorrelated data sets for both M(x) and G(x) (where correlated items had equal preciousness and uncorrelated had varying preciousness). With uncorrelated items, M(x) and G(x) were generally able to reach good solutions in one round for large n. Because of the way M(x) and G(x) are structured, the benefits of sorting the items in descending preciousness are realized with uncorrelated items since the most precious items are put into the solution space in the first round, which enables the algorithms to immediately identify good solutions. This is not the case with correlated items since all items have the same preciousness. Also, the running time of G(1) is much less than M(1) since G(1) is a simple first fit algorithm and M(1) is a sealed bid auction. Therefore, G(x) generally performed better than M(x) with uncorrelated data, except under tighter error constraints.

M(x) performed better than G(x) on correlated data, especially with tighter error constraints (Figure 5.). Test data showed that G(x) is much less likely to converge to a good solution within a limited number of rounds. In

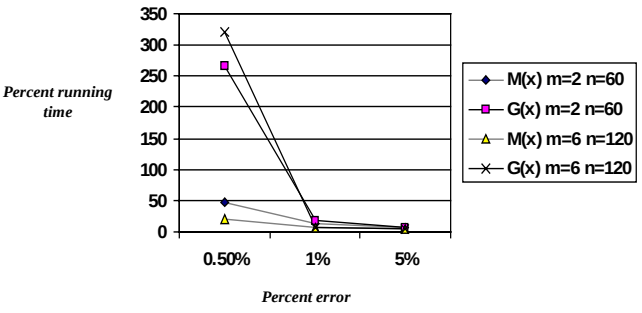


FIGURE 5. AVERAGE PERCENT RUNNING TIMES OF G(X) AND M(X) VS. BB OVER 20 DATA SETS WITH DECREASING ERROR CONSTRAINTS.

general, M(x) performed better than G(x) on data sets where x is greater than [5,10] rounds. This happens when a good solution can't be found by a simple first fit algorithm. M(x) almost always found good solutions in fewer rounds than G(x).

6. Conclusions and Future Work

This paper contributes a novel approach to solving the 0-1 multiple knapsack problem that is simple, robust, and easy to implement. Using agents and markets to model algorithms that approximate the 0-1 multiple knapsack problem works well since it is a resource allocation problem; markets have proven so effective at solving complex resource allocation problems; and it allows for sufficiently efficient implementations. Markets based on auctions permit agents to negotiate with very simple interactions, resulting in very little computational effort, while allowing complex and desirable behaviors to emerge from these simple interactions.

A model is provided that can guide the development of algorithms to solve the 0-1 multiple knapsack problem. One particularly effective and efficient algorithm (M(x)) has been implemented. M(x) uses rounds where agents sell back items, making it effective at finding near optimal solutions. This distinguishes it from simple greedy algorithms by allowing the algorithm to traverse the solution surface. Greedy algorithms stop sooner, but perhaps at a local optimum. The use of agents prevents consideration of infeasible solutions while traversing the solution space, which allows M(x) to traverse sparse solution surfaces, often found with 0-1 multiple knapsack problems, effectively. A computationally simple bidding strategy was chosen for M(x), thus making the algorithm efficient.

The test results confirm that M(x) performs very well. M(x) was generally able to find good solutions in a fraction of the time it took BB to run on larger data sets, and with much less memory. M(x) performed better than G(x) on harder problems and was able to converge to good solutions with fewer rounds than G(x).

```
procedure: M
input: numrounds, sell_random, exchange, knapsacks, items;
output: xstar;
begin
  1. [initialize]
     for i = 1 to m do a_i = size of knapsack i;
     for i = 1 to m do for j = 1 to n do x_ij = 0;
     sort items in descending preciousness p.
     numrounds = x;
     (check exchange ≠ sell_random);
     if sell_random < numrounds then
       numthrowdec=(⌈numthrows /(numrounds / sell _ random)⌉)- 1;
     else numthrows = 1;
     WinningValue = 0;

  2. [Bid]
     while ( numrounds > 0 ) do
       begin
         Value = 0; numrounds = numrounds - 1;
         for j = 1 to n do
           Find ( i: 1 ≤ i ≤ m) where B_i1 ≥ B_l1 > 0
            for all l: (1 ≤ l ≤ m and l ≠ i),
              if no such i exists or if items[1] is stowed then
                   put items[1] at the back of items;
              else x_ij = 1, a_i = a_i - size of items[1],
                   mark items[1] as stowed and move items[1]
                   to the back of items,
                   Value = Value + value of items[1];
         if Value > WinningValue then
             for i = 1 to m do for j = 1 to n do
                 xstar_ij = x_ij, WinningValue = Value;
         [Sell back items]
         call POST(a ,x  Value ,sell_random ,numthrows,
                   numthrowdec, exchange);
       end;
       return xstar;
end.
```

```
procedure: POST
input: a, x, Value, sell_random, numthrows, numthrowdec, kround, exchange;
output: a, x, Value;
begin
  1.  [Sell back random item]
      if (kround%sell_random) == 0 then
          for i = 1 to m do for numthrows / 2 do
              Find (random j:x_il = 1),
                  if such a j exists then
                      x_ij = 0, a_i = a_i + size of items[j], mark items[j] as
                       unstowed, put items[j] in the back of items,
                      Value = Value - value of items[j];
                  else do nothing;
          numthrows = numthrows - numthrowdec;
  2.  [Barter]
       else if (kround%exchange) == 0 then
          for i = 1 to m do for j = 1 to n xtemp_ij = 0
          for numthrows do for i = 1 to m - 1 do
              Find (next j and l:x_ij = 1 and x_(i+1) l = 1 and xtemp_ij ≠ 1
                    and xtemp_(i+1)l ≠ 1),
                    if such a j and l exist then
                       Swap the items, if one of the items does not fit into the
                       Knapsack it was swapped into then mark it as
                       unstowed and put into back of items, Update x, a, and
                       Value, xtemp_ij = 1 and xtemp_(i+1)l = 1;
                    else do nothing;

  3.  [Sell back least precious]
      else
          for i = 1 to m do for numthrows do
              Find ( j: 1 ≤ j ≤ n and x_ij = 1) where p_l ≥ p_j for all l:
                   (1 ≤ l ≤ n and l ≠ j and x_il = 1),
                   if such a j exists then [sell that item back]
                      x_ij = 0, a_i = a_i + size of items[j],
                      mark items[j] as unstowed and move to back of
                    items, Value = Value - value of items[j];
                   else do nothing;

   return x, a, Value, numthrows;
end.
```

LISTING 1. FINAL ALGORITHM

More work could be done in developing more algorithms that use the model from this research, which includes trying new bidding strategies and different market mechanisms. For instance, Park et al. (1999) discuss the trade-offs in a continuous double auction between two extreme bidding strategies where either agents use all relevant information to build internal models on other agents' behaviors before they make bids (e.g. Kreps 1990) or agents require no knowledge about the outside world and do not build models (e.g. M(x)). Exploring more complex bidding strategies (possibly adaptive strategies) may give insight into these trade-offs for this particular problem.

Developing parallel algorithms based on M(x) could yield even faster algorithms. Friedman and Oren (1995) argue that distributed resource allocation can be logarithmic in *m* and linear in *n*, which suggest that parallelization would be possible. It is unlikely that BB can be parallelized since it is a depth-first search algorithm, which is inherently sequential (Reif 1985). It is also unlikely that G(x) can be parallelized, at least completely, since first fit decreasing bin packing is P-Complete[2].

Acknowledgements

I would like to thank the following for their time and suggestions: Dr. Lois Brady, Dr. Hisham Assal, Dr. Terence Critchlow, Dr. Scott Kohn, Nathan Dykman, Dr. Bronis de Supinski, Larry Bolef, and Dr. Len Myers.

References

Aly S. (1994). Object-Agents: A New Role of Design Objects in CAD Systems. in Pohl, J. (Ed.) *Advances in Computer-Based Building Design Systems, 7th International Conference on Systems Research, Informatics and Cybernetics*, Baden-Baden, Germany.

Baker, A. D. (1996). Metaphor or Reality: A Case Study where Agents Bid with Actual Costs to Schedule a Factory. In Clearwater (1996). pp. 184-223.

Chapman, D. (1987). Planning for Conjunctive Goals. *Artificial Intelligence, Vol. 32(3).* pp. 333-377.

Cheng, J. Q. and M. P. Wellman (1998). The WALRAS Algorithm: A Convergent Distributed Implementation of General Equilibrium Outcomes. *Computational Economics, Vol 12.* pp. 1-24.

Clearwater, S. H. (1996). *Market Based Control, A Paradigm for Distributed Resource Allocation.* World Scientific, New Jersey.

Davis, R. and R. G. Smith (1983). Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence, Vol. 20.* pp. 63-109.

[2] See http://www.i.kyushu-u.ac.jp/~seki/P-complete/all/all.html for a list of P-complete problems.

Durfee, E. H. (1988). *Coordination of Distributed Problem Solvers.* Kluwer Academic Publishers, Boston.

Durfee, E. H. and V. R. Lesser (1987). Using Partial Global Plans to Coordinate Distributed Problem Solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* pp. 875-883.

Durfee, E. H. and V.R. Lesser (1989). Negotiating Task Decomposition and Allocation Using Partial Global Planning. In Gasser, L. and M. N. Huhns (Eds.), *Distributed Artificial Intelligence, Vol. II.* pp. 229-243.

Eastman, C., S. Chase, and H. Assal (1992). System Architecture for Computer Integration of Design and Construction Knowledge. *Building Systems Integration Symposium, A/E/C Systems*, Dallas, TX.

Engelbrecht-Wiggans, M. Shubik and r. M. Stark (Eds.) (1983). *Auction, Bidding, and contraction: Uses and Theory.* New York University Press, New York.

Ferguson, D.W., C. Nikolau, and Y. Yemini (1988). Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. *Proceedings of the 8th International Conference on Distributed Computing Systems.*

Ferguson, D. F., C. Nikolaou, J. Sairamesh, and Y. Yemini (1996). Economic Models for Allocating Resources in Computer Systems. In Clearwater (1996). pp. 156-183.

Friedman, E. J. and S. S. Oren (1995). The Complexity of Resource Allocation and Price Mechanisms Under Bounded Rationality. *Economic Theory, Vol. 6.* pp. 225-250.

Gagliano, R. A. and P. A. Mitchem (1996). Valuation of Network Computing Resources. In Clearwater (1996). pp. 28-52.

Garey, M.R. and D. S. Johnson (1979). *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.*

Huberman, B. A. (1988). *The Ecology of Computation.* Elsevier Science Publishers B. V., North-Holland-Amsterdam.

Huberman, B. A. (1995). Distributed Computation as an Economic System. *Journal of Economic Perspectives, Vol. 9(1).* pp. 141-152.

Kirkpatrick, S., C. D. Gelatt, M. P. Vecchi (1983). Optimization by Simulated Annealing. *Science Vol. 220.* pp. 671-680.

Kreps, D. M. (1990). *Game Theory and Economic Modeling.* Oxford; New Yourk: Oxford University Press.

Kurose, J. F. and R. Simha (1989). A Microeconomic Approach to Optimal resource Allocation in Distributed Computer Systems. *IEEE Transactions on Computers, Vol. 38(5).* pp. 705-717.

Kuwabara, K., T. Ishida, Y. Nishibe, and T. Suda (1996). An Equilibratory Market-Based Approach for Distributed Resource Allocation and Its Applications to Communication Network Control. In Clearwater (1996). pp. 53-73.

Malone, T. W., R. E. Fikes, K. R. Grant, and M. T. Howard (1988). Enterprise: A Market-like Task Scheduler for Distributed Computing Environments. In Huberman (1988).

Markland, R. E. (1989). *Topics in Management Science, Third Edition.* John Wiley and Sons, Inc., New York.

Martello, S. and P. Toth (1985). ALGORITHM 632, A Program for the 0-1 Multiple Knapsack Problem. *ACM Transactions on Mathematical Systems, Vol. 11(2).* pp. 135-140.

Martello, S. and P. Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations.* J. Wiley & Sons, c1990.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag.

Marron, D. B. and C. W. Bartels (1996). Computer-Assisted Auctions for Allocating Tradeable Pollution Permits. In Clearwater (1996). pp. 274-299.

Miller, M. S., D. Krieger, N. Hardy, C. Hibbert, and E. D. Tribble (1996). An Automated Auction in ATM Network Bandwidth. In Clearwater (1996). pp. 96-125.

Miller, M. and E. Drexler (1988). Markets and Computation: Agoric Open Systems. In Huberman (1988). pp. 133-176.

Park, S., E. H. Durfee, and W. P. Birmingham (1999). An Adaptive Agent Bidding Strategy based on Stochastic Modeling. *Proceedings of the Third Annual Conference on Autonomous Agents, Seattle, WA.* pp. 147-153.

Pohl, J., L. Myers, A. Chapman, K. Pohl, J. Primrose, and A. Wozniak (1997). Decision-Support Systems: Notions, Prototypes, and In-use Applications. *TR CADRU-11-97,* CAD Research Center, Cal Poly San Luis Obispo, CA.

Reif, J.H. (1985). Depth-first Search is Inherently Sequential. *Inf. Process. Lett., Vol. 20.* pp. 229-234.

Roth, A. E. (1985). *Game-Theoretic Models of Bargaining.* Cambridge University Press, Cambridge.

Sandholm, T. (1993). An Implementation of the Contract net Protocol Modeled on Marginal Calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.* pp. 256-262.

Scarf, H.E. (1984). The Computation of Equilibrium Prices. In Scarf, H.E. and J.B. Shoven (Eds.), *Applied General Equilibrium Analysis.* pp. 415-492. Cambridge University Press, Cambridge.

Steiglitz, K., M. L. Honig, and L. M. Cohen (1996). A Computational Market Model Based on Individual Action. In Clearwater (1996). pp. 1-27.

Stonebraker, M., R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin (1994). An Economic Paradigm for Query Processing and Data Migration in Mariposa. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems.* pp. 58-67. Austin, TX.

Tilley, K. J. (1996). Machining Task Allocation in Discrete Manufacturing Systems. In Clearwater (1996). pp. 224-252.

van Laarhover, P. J. M. and E. H. L. Aarts (1987). *Simulated Annealing: Theory and Applications.* Kluwer Ac. Publ., Dordrecht.

Varanelli, J.M. and J.P. Cohoon (1993). Two-Stage Simulated Annealing. *Proc. 4th ACM/SIGDA Phy. Des. Wksp., Lake Arrowhead, CA.* pp. 1-10.

Walras, L. (1954). *Elements of Pure Economics.* Allen and Unwin. English translation by William Jaffe, originally published in 1874.

Waldspurger, C. A., T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta (1992). Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering, Vol. 18(2).* pp. 103-117.

Wellman, M. P. (1993). A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems. *Journal of Artificial Intelligence Research, Vol. 1.* pp. 1-23.

Wellman, M. P. (1996). Market-Oriented Programming: Some Early Lessons. In Clearwater 1996. pp. 74-95.

Yemini, Y. (1981). Selfish Optimization in Computer Networks. *Proceedings of the 20th IEEE Conference on Decision and Control, San Diego.* pp. 281-285.

Zlotkin, G. and J. S. Rosenschein (1996). Mechanisms for Automated Negotiation in State Oriented Domains. *Journal of Artificial Intelligence Research, Vol.* 5. pp. 163-233.