# Optimization Problems in a Hierarchical Setting

Robert Moll
Bruce MacLeod
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 88-87
October 1988

## Abstract

Many optimization problems in such diverse areas as layout, scheduling, automated assembly, routing and warehousing have an underlying hierarchical structure. In this paper we describe a notation for specifying and solving such problems. The notation has two parts: a static graph-theoretic structure called a *hierarchical constraint graph* or *HCG*; and a control structure called a *policy*. An HCG represents the structure of the relationships that constrain a problem. A policy describes the way in which assignment relationships are to be satisfied and optimized. We have realized this notation in a programming environment which now exists in prototype form. This paper discusses some of the properties of this notation and reports on promising results obtained in the domain of vehicle routing.

# 1 Introduction

Many optimization problems in such diverse areas as layout, scheduling, automated assembly, routing and warehousing have an underlying hierarchical structure. In this paper we describe a notation for specifying and solving such problems. The notation has two parts: a static graph-theoretic structure called a *hierarchical constraint graph* or *HCG*; and a control structure called a *policy*. An HCG represents the structure of the relationships that constrain a problem. A policy describes the way in which assignment relationships that make up the problem are to be satisfied and optimized. We have realized this notation in a programming environment called OPL (Optimization Programming Language) which now exists in prototype form.

An HCG is built from a small set of generalized assignment relationships. These relationships make up the primitives of our notation. They include such relationships as: partitioning – cluster a set of objects into a collection of unordered subsets; and bounded ordering – assign a set of objects to positions in one or several unordered lists. Thus an arc in an HCG represents a stylized assignment between objects associated with a source node and objects associated with a target node. Intuitively, such a relationship can thought of as asserting such assignments as: "if there is sufficient current, plug electrical device A into socket B" or "if there is space, place box A into bigger box B".

HCG machinery gives us a principled decomposition of complex problems into familiar subpieces. Each subpiece - one of our generalized assignment relationships - has the property that well-behaved domain independent algorithms are known. Moreover, the combinatorial optimization technique known as *local search* is applicable to each primitive assignment relationship in a natural way. Local search plays a prominent role in our analysis.

This paper reports on issues that arise when this scheme is applied to three familiar complex optimization problems: a vehicle routing problem; and an extended vehicle routing problem; and a warehouse layout problem.

In section 2 the HCG notation is developed. In section 3 the concepts of improvement policies and improvement programs are introduced. In section 4 we report on OPL's performance on an extended vehicle routing problem. In section 5 we summarize our work and identify directions for future research.

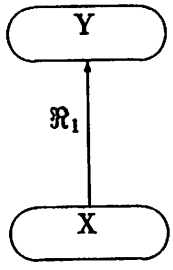
# 2 Hierarchical Constraint Graphs

Generalized assignment relationships are the primitives of the HCG specification system. In this section we describe these primitives, and we present HCG specifications for three example problems: a simple vehicle routing problem, a multiple depot vehicle routing problem, and a warehouse layout problem.

## 2.1 Relations in the Graph

An HCG is a directed acyclic graph that describes the hierarchical relationships among the objects that make up a discrete optimization problem. Nodes in the graph represent a class of objects. An arc from one class of objects to another class indicates that the objects at the tail of the arc are to be assigned to the objects at the head of the arc.

Suppose the set of objects denoted by "X" is assigned to a collection of of objects denoted by "Y". The HCG represents the relationship between the two classes of objects as follows:




We will sometimes write this relationship informally as X→Y.

Frequently it is useful to interpret an assignment primitive as a kind of packing relationship. According to this view, the "Y" objects represent containers and the "X" objects represent components to be placed in the "Y" objects. We will frequently refer to the objects at the head of an arc as *containers* for the *components* at the tail of the arc.

In this paper we will make use of four distinct assignment primitives; additional primitives are easy to add. Definitions of many classes of primitives are sufficiently restricted so that standard approximation and local search procedures have a meaningful interpretation for each primitive type. We describe these four primitives below.

- Partition Relation: a partition relation designates a partition of components among containers. Components in a container are not ordered.
- List Relation : associates each component with a relative position in a list of indefinite length. This relation also partitions components if there are several lists to choose from. There are no fixed positions in which to place a component; only relative postions matter.
- Slotted Relation : associates each component with a position in one of several "slotted" containers, each of which has a fixed number of slots. This relation also partitions components if there are several containers to choose from.
- Ring Relation : associates each component with a position in one or several ordered rings of indefinite length. Thus this relation also induces a partition on components if there are several rings to choose from.

Each of the above relations can be augmented with additional constraints of various kinds. We give some examples below.

- container constraints: a component may be constrained to fit in some subset of the possible containers.
- capacity constraints: each component may have a fixed size, and containers may have capacity constraints on the summation of component sizes.
- position constraints: a component may have restrictions on its position in a container.
- relative position constraints: a component may have constraints on its position in relation to other components in a container.

Finally, we allow a portion of a primitive assignment relation to be initialized when an HCG is first created.

Thus, a node in an HCG represents a class of objects; an arc represents the constraining relationship between the objects at its source and target nodes. A problem is specified in HCG notation by first identifying the constituent classes of objects that make up the problem and then identifying and, if neccessary, initializing, the primitive relationships among these classes.

## 2.2 Assignments in the HCG

A feasible solution to an optimization problem that has been specified in our notation is obtained when all assignment relationships in the problem's HCG have been satisfied for every object. In OPL relationships are satisfied incrementally by application of well behaved approximation algorithms which are associated with the assignment primitives. Local search routines are also part of the solution process. They too are associated with the primitive assignment relationships of the notation and they, too, are largely domain independent. A typical solution method for a problem instance combines approximation and local search algorithms to achieve a satisfactory solution. We describe our primitive approximation and local search algorithms in this section. Later sections address the manner in which approximation and local search procedures can be combined to solve a particular class of problems.

Typically our primitive approximation procedures closely resemble the family of approximation algorithms available for bin packing problems. Thus for each primitive relationship we formulate algorithms which we identify as *first-fit*, *best-fit*, and *next-fit*. Each algorithm is supplied with: an order in which containers (and positions in the container) are considered; and (especially in the case of best-fit) a metric that evaluates a particular assignment.

Thus *first-fit* applied to the slotted container primitive considers each of the containers and each slot in each container in some order, and makes the indicated assignment to the first slot that can legally "hold" the object being assigned.

In the case of the ring relationship the bin packing model is less appropriate. Here, for example, we might interpret best-fit to mean an algorithm that places an object on the ring next to the object that it is closest to, according to some predefined notion of distance.

Local search procedures are also associated with our primitive relationships in a natural way. For example, the classical *2-Opt* procedure of [Kernighan and Lin 1973] has a natural interpretation for the ring relation.

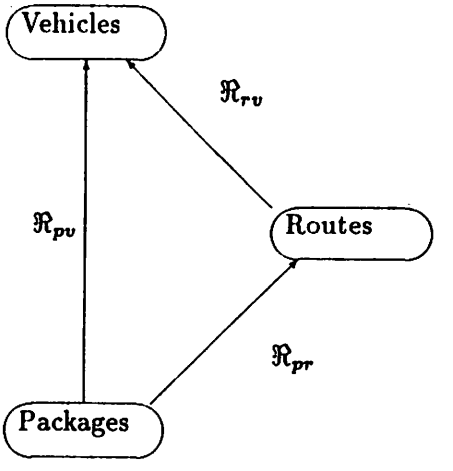
For the parition relation, we make use of a simple 2-Swap routine, which has the following interpretation: exchange two elements from different partitions, and check for improvement. Similarly, 2-Swap for slotted containers means: exchange entries in adjacent slots, and check for improvement. Our local search methods can easily be extended to 3-Opt, k-Opt, k-Swap, etc.

In the next section we use our specification machinery to define the structural characteristics of several common discrete optimization problems.

## 2.3 An HCG Definition of the Vehicle Routing Problem

In the Vehicle Routing Problem (VRP), vehicles deliver packages to a collection of geographically dispersed customers. Each vehicle has a carrying capacity and possibly a maximum travel time. The goal of a VRP instance is to find a solution that respects all constraints and at the same time minimizes the total travel times of the vehicles. This is the simplest of a large and complex family of vehicle routing problems. Later we will discuss a more complicated version of the problem.

To formulate this problem in HCG notation we recast the fundamental relationships of the problem using our primitive assignment relations. The graph below identifies these primitives and their interrelationships.




where

- $\Re_{pv}$ partitions packages among vehicles. The position of a package in a vehicle is not important (though it might be for some variants of the VRP), so $\Re_{pv}$ can be represented by the partition relation primitive.

Weight capacity constraints are considered to be part of the primitive relationship.

- $\Re_{pr}$ associates packages with a position in some route. The route is represented by a ring relationship. The ring begins at the depot; the route is the (clockwise) natural one beginning at the depot.

- $\Re_{rv}$ associates each route with a single vehicle. The route travel time is computed as a function of the packages assigned to the route. The vehicle travel time capacity must be respected with each assignment. For each route $\Re_{rv}$ is initialized to include the single dispatch depot of the problem as its first entry. $\Re_{rv}$ can be thought of as a special case of a partition relation.

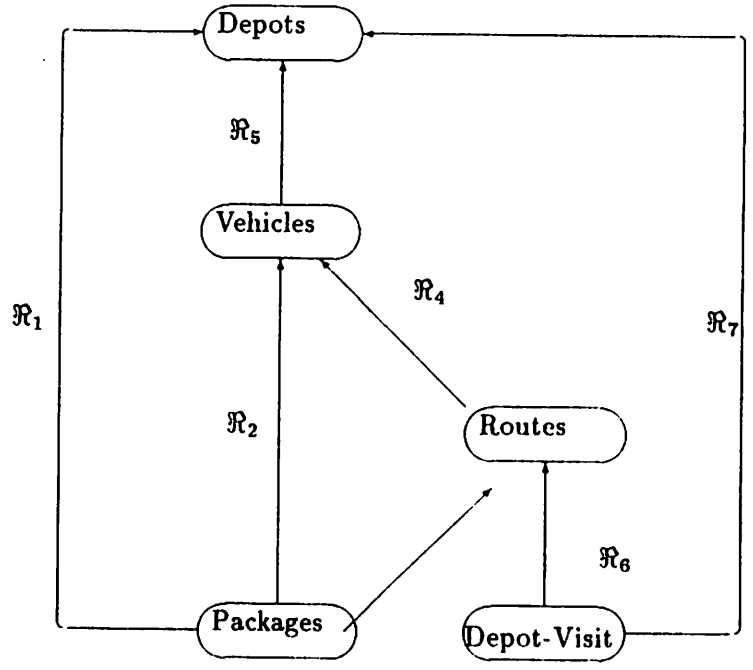
The HCG for the VRP is a *commutative* graph, by which we mean that the transitive closure of the assignment relationships of the graph must be consistent. Here commutivity means that if a package is assigned to a vehicle and is also assigned to a route, then that route must be assigned to the vehicle that holds the package. In this work we restrict our attention to commutative HCGs. Commutivity is a minor restriction on the power of HCGs as a specification system, and at the same time it is an important computational asset.

By defining the VRP in terms of a fixed set of simple assignment relations, domain independent approximation and local search routines can be applied to the pieces of this VRP problem to yield well-behaved solutions. These solutions are discussed in section 3.

## 2.4 An HCG Definition of the Depot Vehicle Routing Problem

We next consider a somewhat more complex variant of VRP in which there are multiple depots (DVRP). Vehicles are assigned to one of several depot and each customer delivery is made from one of the depots.

The HCG for this problem is given below:




where

- $\Re_1$ partitions packages among depots. Since the position of a package in a depot is not important, $\Re_1$ can be represented by a partition relation. If capacity for packages in a depot constrains this assignment, then the relation will include this constraint.

- $\Re_2$ partitions packages among vehicles. The position of a package in a vehicle is not important (although it might be for some variants of the VRP), so $\Re_2$ can be represented by a partition relation. Any weight or size limits are included in the constraint of the relationship.

- $\Re_3$ associates packages with a position in some route. As before a route is represented by a ring relation.

- $\Re_4$ associates each route with a vehicle. The route travel time is computed as a function of the packages assigned to the route. The vehicle travel time capacity is respected in each assignment. If there are multiple routes configured to the vehicle, the order in which each route is visited may be important, in which case $\Re_4$ is a list relation, or unimportant, in which case $\Re_4$ is represented by a partition relation.

- $\Re_5$ partitions vehicles among depots. There may be limits on the number of vehicles at a depot and the limits may be a function of vehicle characteristics; these limits are represented by capacity constraints in the partition relation.
- $\Re_6$ associates one depot visit with each route. Each route must start from some depot and return to the depot; this partition relation models the depot visit and has a capacity for only one depot visit per route.
- $\Re_7$ trivially assigns each depot-visit to its depot.

This problem can be viewed as a series of separate VRP problems in a natural way once packages and vehicles have been assigned to depots.

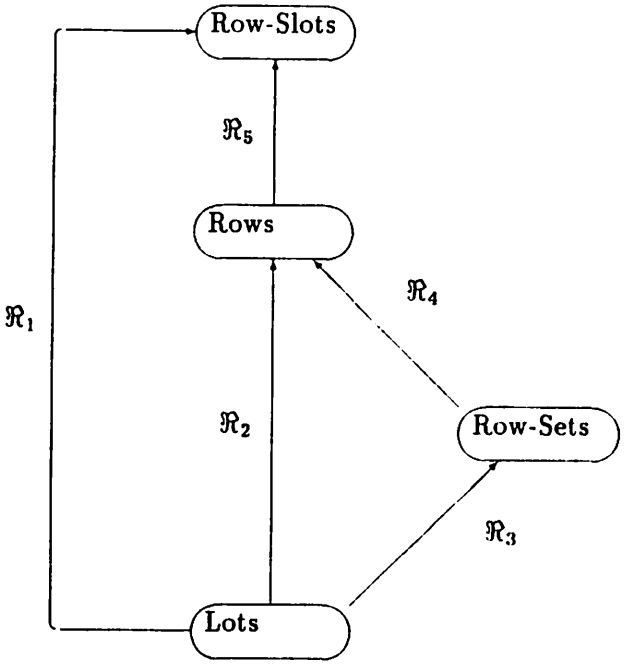
## 2.5 An HCG Definition of a Warehousing Problem

The HCG notation is applicable to a broad class of problems. In particular, problems involving warehouse layout and check processing [Markland and Nauss 83], [Macleod 89] have been represented using this notation. To illustrate the broad applicability of the notation, we include a description of a simple warehouse layout problem.

A collection of items, grouped into lots, are to be stored in a warehouse. Rows of bins are located on a warehouse floor and lots are to be placed in the bins. Because orders for lots have already been received, information on the frequency with which lots appear on the same order has been tabulated in an affinity matrix. A high score in cell i-j of the matrix indicates that lot i and lot j appear frequently on the same orders. The goal of the warehousing problem is to assign lots to bins in such a way that the total travel distance incurred as all orders are filled is minimized.

In addition to the adjacency preferences between lots, other considerations may also affect the layout of the warehouse. For example, if there is only one door to the warehouse, then there will be some advantage to placing high demand lots near the door.

Informally, we will solve the problem by first grouping lots into sets in which the members have common affinity. Then we order each grouping individually, in effect placing the lots in rows without positioning them on the floor. Finally we place the rows into "row slots" on the floor–that is, we order the rows. Along the way, we will apply various local search algorithms to improve the current partial solution. The HCG for this representation is given below:




where

- $\Re_1$ partitions lots to row slots on the warehouse floor. Row slots represent the fixed "width of the warehouse" positions on the warehouse floor where rows can be placed.
- $\Re_2$ associates a lot with a position in a row. Since the number of positions in a row is fixed, this relationship is best represented by a slotted relation.
- $\Re_3$ partitions lots into groups with high mutual adjacency preferences. Lots in this relation are unordered.
- $\Re_4$ associates a row-set with a particular row. Since there a fixed number of slots in the row, a capacity constraint is needed to restrict the number of lots in a group to be less than or equal to the number of bins in a row.

- $\Re_5$ associates a row with a position on the floor. A row can fit into a fixed number of positions on the warehouse floor, so this relationship is best represented by a slotted relation.

The above example could be extended to more complicated variants. For example, if the warehouse has several floors, then a new floor node could be added, along with arcs to this node from the lots, rows, and row-slots, and the floor node.

# 3 Policies for Assignment and Improvement in the Hierarchical Constraint Graph

The HCG clarifies the static relationships among objects of a hierarchically structured optimization problem. A feasible solution has been constructed when all static relationships are satisfied for each object in the problem space. In this section we describe our method for constructing solutions to problems that have been posed using HCG notation. The key primitive of the solution method is called an *improvement policy.*

An improvement policy seeks to place or rearrange a collection of objects in such a way that the new solution extends or improves the old solution. More concretely, an improvement policy is a triple consisting of:

- a collection of objects, called *primary objects*;
- a sequence $(t_1,...,t_k)$ of transformations called an *improvement sequence*, where each $t_j$ is an approximation or local search routine which is associated with a particular HCG arc; and
- a collection of HCG arcs called *bound* arcs, which identify those relationships in an existing partial solution that may not be altered by the improvement sequence. Arcs which are not bound are called *free* arcs.

A *policy program* is a program that has improvement policies as primitives, and includes in addition elementary programming constructs (if, do-while, for, ..) and some data structures functions (sorting, extracting, merging, ...). The notions of improvement policy and policy program are developed in this section. We begin with two informal examples that describe policy programs for VRP.

## 3.1 A Vehicle Routing Problem Instance

In this section we consider two VRP examples. Our examples are based on the HCG presented in section 2.3

In the first example we assume that vehicles are identical and that vehicle capacity for packages is limited in the sense that no single vehicle can carry a significant fraction of the packages. (This capacity constraint is considered to be an implicit part of the relation $\Re_{pv}$ and is identified when $\Re_{pv}$ is first specified.) We assume that there are no limits on vehicle tour distance. The cost of a solution is the total distance traveled by the vehicles.

Informally here is our solution. First we identify $\Re_{pr}$ as an example of the ring data structure. It is initialized to include the dispatch location of the vehicles as its first entry. Then we associate an angle with each package delivery site, using the dispatch depot of the vehicles as origin. We sort packages by angle and then assign them to vehicles –a partition relationship– on a "next-fit" basis. That is, we attempt to place a package in the current vehicle, and if it won't fit, we place it instead in the next vehicle that is completely unoccupied. This assignment crudely clusters packages with nearby destinations into the same vehicle. Next we route each vehicle: we examine each package in a vehicle, in turn, and insert it in that vehicle's route in a position that leaves it closest to a package that has already been placed in the route. Once these routes have been constructed we apply 2-opt local search to optimize each route.

Our algorithm does a reasonable job of solving this simple version of VRP. Below we describe how we build a policy program that realizes this algorithm. We construct this policy program in four steps.

- **Step One:**

  The first step establishes the packages to vehicles assignment. We accomplish this by sorting packages by polar angle, as described above. Such sorting routines are a permissible primitive of the policy program language. Next we apply our first improvement policy, which we call $IP_1$. The improvement sequence of $IP_1$ consists of a single approximation routine. The $\Re_{pv}$ relation is a partition relation, and we perform the partition using the following next-fit algorithm:

  next-fit($\Re_{pv}$,all-containers,t)

Here the second parameter identifies the subset of legal containers (vehicles) to place the components (packages), and the third parameter identifies a measure that is used to evaluate the quality of alternative assignments. In this case, the metric always returns a true value and therefore the placement is always accepted if a feasible fit exists (the constraints of the relation are not violated). Thus, $IP_1$ is a bona-fide policy consisting of a transformational sequence with one entry (the call to next-fit), a domain of primary objects consisting of all packages, and an empty set of bound arcs. We summarize step one as follows:

packages = sort(PACKAGES,polar-angle)
$IP_1$(packages)

If any packages are unable to be placed in a vehicle, they are assigned to the package's *leftover list*. There is a leftover list for every node in the HCG. We discuss its role later in this section.

- **Step Two:**
  In step two, improvement policy $IP_2$ is applied in an attempt to route the packages associated with each vehicle. The algorithm considers each package in a vehicle and places that package in the best possible position in the route. The improvement sequence is again one approximation procedure:

  best-fit($\Re_{pr}$,all-containers,least-travel-distance)

  Packages will be assigned to a position in a route which incurs the least additional travel distance. The underlying OPL machinery will automatically maintain consistency in the following sense: packages in different vehicles cannot be assigned to the same route.

  Steps one and two together yield the following policy program:

```
packages = sort(PACKAGES,polar-coordinates)
IP1(packages)
For each vehicle in VEHICLES
    begin
        packages = packages in vehicle
        IP2(packages)
    end
```

$IP_2$ will be called as many times as there are vehicles. Each time it is called, the packages belonging to the vehicle under consideration are designated as the primary objects.

- **Step Three:**
  $IP_3$'s singleton improvement sequence improves upon the existing partial configuration by performing local search on each of the routes using 2-opt local search. The packages to vehicles and packages to routes arcs are bound.

  2-opt($\Re_{pr}$,packages-in-same-route,travel-distance)

  For efficiency, the second parameter of the 2-opt call guarantees that only packages in the same vehicle will be considered for 2-opt exchanges. Exchanges across vehicles can never be successful because the packages to vehicles arc is bound. The functional metric that judges the quality of an interchange is given in the third parameter.

  Our policy program is expanded to include step three:

```
PACKAGES = sort(PACKAGES,polar-coordinates)
IP1(PACKAGES)
For each vehicle in VEHICLES
    begin
        packages = packages in vehicle
        IP2(packages)
        IP3(packages)
    end
```

- **Step Four:**
  $IP_4$ attempts to satisfy the remaining relation, $\Re_{rv}$. Since the packages in a route have already been assigned to a vehicle, each route is bound to the vehicle that its packages are assigned to. In addition, it has been assumed that no travel time constraints are associated with this relation. Thus, the required assignment has been made implicitly already, and we make the assignment explicit using the following pol-

icy:

first-fit($\Re_{rv}$,all-containers,t)

The underlying OPL machinery only allows the single feasible vehicle to be considered for each route. This vehicle satisfies the metric (always true) and the route is assigned to it.

The preceding steps can be combined to form following policy program.

```
package = sort(PACKAGES,polar-coordinates)
IP1(packages)
For each vehicle in VEHICLES
    begin
        packages = packages in vehicle
        IP2(packages)
        IP3(packages)
    end
IP4(ROUTES)
```

The policy program above can now be applied to the problem instance. The program may also be augmented in order to obtain a (hopefully) improved solution. One such improvement policy would attempt to improve the tour length by interchanging packages between routes (and vehicles) according to a suitable metric before applying 2-opt local search to individual routes. This policy would involve having two primitive improvement procedures operate together in order to improve the solution to a problem. Our second example illustrates this cooperative process in a more elaborate way.

In the second example we assume, as before, that vehicle capacity is limited. In addition we assume that each vehicle has a maximum allowable tour length. Once again these constraints are posed as part of the initial specification for the HCG relations. To handle this new situation we develop a single policy which deals with both constraints before making final assignments. That is, we consider each package in turn; if a package fits in a vehicle and fits in the route associated with that vehicle without making the route too long, then the package is assigned both to the vehicle and to the route. Let $IP_5$ denote an improvement policy with the following (slightly abbreviated) improvement sequence:

(next-fit($\Re_{pv}$) best-fit($\Re_{pr}$) first-fit($\Re_{rv}$))

Here $\Re_{pv}$ limits the number of packages assigned to vehicles (based on vehicle capacity) and $\Re_{rv}$ constrains the assignment of routes to vehicles by invalidating tours that exceed the maximum allowable tour length.

Unlike the previous example, which applied an approximation or local search primitive in isolation from other primitives, these three approximation procedures work together. For a given primary object–in this case, a package – all three primitive approximation procedures must be satisfied for a (three-part) assignment to occur.

To describe this cooperative process, let $p_t$ denote a package which is under consideration. Next-fit($\Re_{pv}$) is the first approximation procedure to be applied to $p_t$. Next-fit operates in the context of previous assignments, so the last container (vehicle) to be considered is evaluated first. If there is a successful fit, then the remaining two relations are applied. If a fit to the latest vehicle is not successful (capacity constraints are violated), then the next vehicle in the list of vehicles is considered. Suppose next-fit($\Re_{pv}$) establishes $v_i$ as a feasible vehicle for $p_t$.

A tentative assignment exists between $p_t$ and $v_i$. This assignment is subject to successful assignments by best-fit($\Re_{pr}$) and first-fit($\Re_{rv}$). The next entry in the improvement sequence, best-fit($\Re_{pr}$), is applied first. The only routes that will satisfy the transitive properties of the HCG are those that are assigned to $v_i$; in this case, we assume at most one route $r_i$ is assigned to $v_i$. Best-fit($\Re_{pr}$) establishes a position in the ordered ring of $r_i$ for $p_t$ which results in the least distance traveled.

The assignment of $p_t$ to a position in $r_i$ is only a tentative one; the remaining procedure, first-fit($\Re_{rv}$), must be satisfied. $\Re_{rv}$ establishes an assignment of routes to vehicles. The question of which route to assign to a vehicle is resolved automatically by configuring those objects which are "related" to the object $p_t$; the route $r_i$ is the container for $p_t$ so first-fit($\Re_{rv}$) is applied to $r_i$. Recall that this assignment is no longer a trivial one: it is constrained by the tour length capacity associated with vehicle $v_i$. If $r_i$ has been previously assigned to $v_i$ – an assignment that might have been induced by an earlier package assignment– then the configuration must be checked again because $p_t$ has been added to $r_i$.

Suppose the configuration of $r_i$ to $v_i$ is invalid: the tour length is too long. When this occurs, OPL backs up to the previous entry in the improvement sequence and returns failure on the tentative assignment. The assignment of $p_t$ to $r_i$ failed because of the failed $r_i$ to $v_i$ assignment, so the next best route must be considered in $\Re_{pr}$. But assuming $p_t$ is not the first package

in the vehicle, the only legal routes are those that are assigned to $v_i$. Since $r_i$ is the only route assigned to $v_i$, the best-fit($\Re_{pr}$) has run out of routes to consider. So best-fit($\Re_{pr}$) returns failure and the assignment of $p_t$ to vehicle $v_i$ is considered invalid, causing next-fit($\Re_{pv}$) – the first entry in the improvement sequence – to place $p_t$ in $v_{i+1}$, the next vehicle in the vehicle list. Then the improvement sequence attempts to deal with $v_{i+1}$ in the same manner as $v_i$.

The policy program that includes $IP_5$ is straightforward:

packages = sort(PACKAGES,polar-angle)
$IP_5$(packages)

The above solution procedure could be modified to make the the assignment of a package to a route more efficient. In particular $p_t$'s tentative assigned route, $r_i$, could be optimized using local search before the $r_i$ to $v_i$ assignment is attempted. This would result in the following improvement policy:

(next-fit($\Re_{pv}$) best-fit($\Re_{pr}$) 2-opt($\Re_{pr}$) first-fit($\Re_{rv}$)).

## 3.2 Discussion

An improvement policy applies approximation and local search routines to realize the assignment relationships specified by the HCG. The improvement sequence portion of a policy identifies and orders a collection of such routines; the sequence can be used to realize and optimize one or several assignment relations in the graph. Applicability of local search is controlled by designating certain arcs in the HCG as bound arcs; such arcs represent assignments which may not be extended or altered.

The improvement policy construct provides a method for dealing with the mix of optimization and constraint criteria that can arise in a complex optimization problem. Thus in the second example of the last section we describe an optimization criterion– minimize total tour length – constrained by a route length bound for each vehicle as well as a vehicle capacity bound. Because vehicle capacity and tour length are interrelated, the three assignment relationships $\Re_{pr}$, $\Re_{pv}$, and $\Re_{rv}$ must be addressed at the same time.

As we have seen, an improvement sequence that applies routines to several HCG arcs adds flexibility to the notation by allowing interrelated con-

straining relationships to be addressed. At the same time, such a construction can incur considerable computational overhead. If the final routine in a multiple entry policy fails to meet the constraints set out in the problem specification, OPL must backtrack through the improvement sequence in search of a collection of composed assignments that will indeed respect all constraints. This phenomenon is probably unavoidable in complex problems with many interdependent constraints. The inclusion of conditional constructs in the notation will, hopefully, alleviate some of these difficulties. Thus, the conditional structure of a policy program can be used to channel the solution to the policies that do the best job of meeting the difficulties presented by the problem data.

The size and complexity of many optimization problems means that a solution must be developed in stages. The decomposition induced by an HCG specification and exploited by the improvement policy mechanism provide a principled way to develop such staged solutions.

### 3.2.1 Efficency Considerations

Formal properties of improvement policies and properties of HCGs can be used to improve the efficiency of OPL computations. In this section we discuss several of these properties.

The first property we exploit to achieve more efficient computations is the transitivity of the HCG. Frequently alternative assignments of an object to a container can take place along different paths in the HCG graph. These alternative assignments must be consistent with previously made assignments. OPL always checks alternative paths before applying an approximation routine. If an alternative path already contains an assignment for the relevant object, then this assignment is fixed and no significant processing need be done.

OPL achieves further efficiencies through the use of a principle we term *partition invariance*. We illustrate this principle with an example from the DVRP problem of section 2.4. Suppose that as part of a local search routine we wish to exchange packages between two distinct vehicles. The capacity constraints of the vehicles must be checked before such an exchange is accepted, and so too must any routing constraints associated with the two vehicles. However if depot package capacity constraints are present, they need to be checked *only* if the two vehicles involved in the exchange are assigned to different depots. If they are assigned to the same depot, then then the depot package capcity constraint is invariant under the exchange.

OPL detects such invariants automatically and avoids constraint checking when possible.

The *bound arc consistency principle* leads to a third efficiency. Consider the following DVRP example. Suppose two packages are to be interchanged between depots, and suppose the relation package→vehicle is bound. If one of the packages is not the sole occupant of its vehicle, then the proposed exchange of packages between depots can never be productive: Changing depots will force a package to be assigned to a new vehicle, thus violating the bound status of the package→vehicle relation. OPL detects such implicit inconsistencies automatically and eliminates the resulting unneeded processing.

### 3.2.2 Leftover Processing by Local Search Procedures

Incompletely configured objects are an integral part of OPL. Incomplete assignments can occur, for example, when improvement policy reaches an bottleneck in the placement of objects. That is, a grouping or partition may force an unsatisfiable assignment in later stages of processing. As a result, a solution may be incomplete after application of a policy or a policy program, and certain objects may appear on leftover lists.

Our approach to this phenomenon is to broaden our view of a "legal" solution. We simply assume that objects may appear on leftover lists, and we rely on our existing machinery to attempt to remedy the situation. Thus, our approach is to permit leftover objects to be identified as primary objects which are eligible "domains" for improvement policies. Of course the form of local seach applied to such primary objects differs from the local search machinery we have seen so far. For example, we make use of such routines as External-1-Swap, which simply places a leftover object in some legal position; and External-2-Swap, which exchanges a fully assigned object for a leftover, in the hope that the new leftover will be easier to place by External-1-Swap. In [Moll,Healy, 88] we demonstrate the efficacy of this view of local search for rectangle packing.

Such machinery is in place in the current OPL prototype. We report on one application of leftover processing in Section 5.6.

## 4 OPL and the DVRP Problem

In this section we evaluate the effectiveness of the OPL system by comparing policy program results to previously published results. In particu-

lar we consider three DVRP data sets taken from the literature, and we construct a single policy program that significantly outperforms published results on these data. We also examine policy program behavior on an especially tightly constrained data set, and we show how OPL can be made to perform reasonably on such a data set.

The policy programs we develop have the following general form: first, a relatively good approximate solution is developed; then, local search improvements are made to that approximate solution to achieve a final solution. Within this general setting we consider a variety of methods for achieving approximate solutions and for optimizing these solutions using local search. In general, these experiments indicate that if problem constraints are not particularly significant, then the final solution is independent of the feasible solution method. In problems with significant constraints, however, policy programs which apply local search to intermediate partial solutions are particularily effective in solving the DVRP.

Finally, we develop and evaluate policy programs for a DVRP instance which is tightly constrained. One of the three datasets is modified to include packages and vehicles of different sizes and constraints on route distance. In addition, capacity constraints for depots, vehicles, routes play a significant role in restricting feasible solutions. Results from this analysis provide additional support for our observations about constrained problems and also demonstrate one method of handling leftovers.

## 4.1 A Benchmark of the OPL Environment

While there are many algorithms for the DVRP, few include test data. Datasets are included in the work of [Gillett and Johnson, 1974], [Gillett and Johnson, 1976], [Perl and Daskin,1985], and [Perl, 1987]. The research of Gillet and Johnson adapts the single depot sweep algorithm to the multiple depot problem. In addition they incorporate the 3-opt and variable-depth local search procedures of Lin and Kernighan. Perl and Daskin solve both the depot location and the multiple depot routing problem. Their algorithm for DVRP involves application of the savings method to develop a route and then application of two improvement procedures. Specifically, they apply a single site exchange local search procedure and the 2-opt local search procedure of Lin and Kernighan. We chose to use the three datasets from the recently published results of [Perl and Daskin,1985] and [Perl, 1987].

We begin with a subjective characterization of the three datasets so as to better understand the effect of different policy programs. Dataset 1 can be

characterized as the "purest" of the three datasets. There are 55 packages, 3 depots, and 10 vehicles. All packages have a size of 20 units, all vehicles have a capacity of 120 units, and at most 27 packages can be placed at any one depot. The only constraint that is particularily restrictive is the vehicle capacity for packages. Dataset 2 is also relatively "pure". There are 55 packages, 4 depots, and 12 vehicles. All packages have a size of 20 units, all vehicles have a capacity of 100 units, and at most 19 packages can be assigned to any one depot. Both vehicle and depot capacity significantly restrict problem solutions. Dataset 3 is the largest of the three. There are 85 packages, 3 depots, and 18 vehicles. All packages have a size of 20 units, all vehicles have a capacity of 100 units, and at most 39 packages can be assigned to a depot. Once again both depot and vehicle capacity are significant constraints. Table 1 summarizes our subjective characterization of the three datasets.

| | DataSet 1 | DataSet 2 | Dataset 3 |
|---|---|---|---|
| $\Re_{pd}$: packages→depots | weak | moderate | high |
| $\Re_{pv}$: packages→vehicles | moderate | high | high |

Table 1: Subjective Characterization of Constraints

We applied a number of policy programs to these data sets. Below we describe the policy program that achieves the best results on the three DVRP datasets. Later we will summarize the effect of other policy programs on the three DVRP datasets.

Table 2 outlines the policy program with the best performance; we have split this program into two parts which are separated by the double line in the figure. The first part establishes and initial feasible solution and the second part optimizes this solution. (The tables in this section present policy programs in a somewhat simplified form.) In general terms, the structure of this program can be described as follows. First, packages are assigned to the closest depot that has sufficient available space. The first improvement policy of policy program I implements this assignment (a bullet in the table indicates either an improvement policy or another primitive of the policy program). Next, pairs of packages are swapped (2-Swap) between their assigned depots if the swap reduces total distance from package site to assigned depot.

| (Value of Reported Results) | DataSet 1 (2263) | DataSet 2 (428.95) | Dataset 3 (689.85) |
|---|---|---|---|
| **Policy Program I** | | | |
| • Best-fit($R_{pd}$,distance-to-depot) | | | |
| • 2-Swap($R_{pd}$,all,total-distance-to-depots) | | | |
| ◦ Sort by Angle from Assigned Depot | | | |
| • Next-Fit($R_{pv}$,) | | | |
| ◦ First-fit($R_{pr}$,) | | | |
| ◦ Satisfy remaining relations | | | |
| • 2-Opt($R_{pr}$,route-neighbors,distance) | 2532 | 464.8 | 720.7 |
| ◦ 1-Move($R_{pr}$,all,distance) First-Fit($R_{pv}$,) | 2271 | 446.3 | 714.7 |
| • 2-Swap($R_{pr}$,all,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2254 | 441.5 | 693.4 |
| • 2-Opt($R_{pr}$,route-neighbors,distance) | 2253 | 441.0 | 688.2 |
| • 2-Reinsert($R_{pr}$,all,distance) First-Fit($R_{pv}$,) | 2247 | 427.5 | 681.7 |
| ◦ 1-Move($R_{pr}$,all,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2126 | 418.6 | 681.7 |
| • 2-Swap($R_{pr}$,all,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2126 | 418.6 | 672.2 |
| • 2-Opt($R_{pr}$,route-neighbors,distance) | 2121 | 418.6 | 671.4 |
| ◦ 2-Reinsert($R_{pr}$,all,distance) First-Fit($R_{pv}$,) | 2095 | 418.6 | 671.4 |

Table 2: Best-Performing Policy Program

The next three steps implement the sweep-fit procedure which is described in detail in the introductory example. After completing the assignments of sweep-fit, the relations $R_{vd}$ and $R_{rv}$ are as yet unsatisfied. However, transitivity considerations limit the assignment of each route and vehicle component to only one container. Since there are no constraints on these relations, we denote the resolution of $R_{rv}$ and $R_{vd}$ assignments in Table 2 by the term "Satisfy remaining relations".

Finally, since packages are assigned to the first feasible position in a route, the route assignments are improved with the "2-Opt" local search function. This completes the description of the first part of policy program I. Policy program I achieves a feasible solution value which is within 12% of the reported results for the three datasets, which are reported in parentheses at the top of each column.

Optimization occurs in the second part of policy program I. The general strategy is to address potential problems in the feasible solution with a collection of improvement policies. This collection of policies includes: move a package from one route to another (1-Move), interchange the positions of two packages (2-Swap), remove two packages from their routes and reinsert them in the best possible position in each other's route (2-Reinsert), and the two-opt procedure of Lin and Kernighan (2-Opt).

The effective search neighborhood for a policy is constrained by the bound relations associated with that policy. For example, the first improvement policy in the second part of program I (1-Move) has two free relations $R_{pr}$ and $R_{pv}$; the remaining relations are bound. This restricts 1-Move to movements of a package within the same depot. In the fifth improvement policy, 1-Move is again applied, but in this case the effective neighborhood is not limited to the assigned depot.

The results in Table 2 demonstrate the effectiveness of the policy program construct. The approximation algorithm of Perl and Daskin is improved upon by 7.4%, 2.4%, and 2.8% respectively for the three datasets. Apparently, the two 1-Move policies of program I are in large part responsible for the substantial reduction in routing distance that occurs for dataset 1. 1-Move allows a package to move to a new vehicle (or depot in the second 1-Move) if the overall routing distance can be decreased. The effect of 1-Move is to override the initial assignment to the closest vehicle (or depot). A subjective characterization of dataset 1 indicates that this is a relatively pure, unconstrained instance of the DVRP, so the results of 1-Move can be interpreted in the context of very loose capacity constraints. The effect of 1-Move on the other two datasets is not as dramatic; apparently depot and

vehicle constraints limit the movement of packages.

The results of this section demonstrate that the improvement policy mechanism can be an effective approach to solving hierarchically structured discrete optimization problems. The next section takes a closer look at the relative performance between policy programs in order to determine why some did better than others.

## 4.2 A Comparative Evaluation of Policy Programs

In this section we address several issues as they relate to the DVRP problem:

- How important is the application of local search at intermediate stages in the construction of problem solutions? How sensitive is the solution to the form of local search applied?
- Does the order of application of local search improvement policies affect the final solution value ?

### 4.2.1 Local Search Applied to Partial Solutions

In order to gain insight into the importance of local search for intermediate solutions we constructed and tested two variants of the policy program reported on in table 2. The first, policy program (II) is identical to the table 2 program (I) in all ways except that the intermediate local search (2-Swap) of that program is not applied. The results of that experiment are reported in table 3.

The results in table 3 show that local search applied to a partial solution has no effect for datasets 1 and 2. In dataset 3, intermediate local search does play an important role. Without it, the final solution value of policy program II is 12% above the winning policy program solution value. These results make sense in the context of Table 1. For dataset 3, the assignment of packages to depots is tightly constrained and packages cannot always reside in their depot of choice. After a complete initial solution is developed, 1-Move has little flexibility and 2-Swap cannot perform a redistribution of packages between depots.

While policy program II eliminates an intermediate local search routine from program I, program III varies program I by substituting a different intermediate local search routine for the 2-Opt routine in the first part of the program. Table 4 summarizes the effect of this strategy. Policy program III first assigns packages to their closest depot. Then a 1-Move local search

| (Value of Reported Results) | DataSet 1 (2263) | DataSet 2 (428.95) | Dataset 3 (689.85) |
|---|---|---|---|
| **Policy Program II** | | | |
| • Best-fit($R_{pd}$,distance-to-depot) | | | |
| ◦ Sort by Angle from Assigned Depot | | | |
| ◦ Next-Fit($R_{pv}$,) | | | |
| • First-fit($R_{pr}$,) | | | |
| ◦ Satisfy remaining relations | | | |
| • 2-Opt($R_{pr}$,route-neighbors,distance) | 2532 | 464.8 | 786.3 |
| • 1-Move($R_{pr}$,all,distance) First-Fit($R_{pv}$,) | 2271 | 446.3 | 780.2 |
| ◦ 2-Swap($R_{pr}$,all,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2254 | 441.5 | 765.0 |
| • 2-Opt($R_{pr}$,route-neighbors,distance) | 2253 | 441.0 | 764.5 |
| ◦ 2-Reinsert($R_{pr}$,all,distance) First-Fit($R_{pv}$,) | 2247 | 427.5 | 755.6 |
| • 1-Move($R_{pr}$,all,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2126 | 418.6 | 754.8 |
| • 2-Swap($R_{pr}$,all,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2126 | 418.6 | 754.8 |
| • 2-Opt($R_{pr}$,route-neighbors,distance) | 2121 | 418.6 | 754.8 |
| ◦ 2-Reinsert($R_{pr}$,all,distance) First-Fit($R_{pv}$,) | 2095 | 418.6 | 754.8 |

Table 3: Policy Program without Intermediate Local Search

procedure is applied in order to move packages to another depot if the closest adjacent package does not reside in the current depot.

The results from this strategy are almost uniformily bad. Only for dataset 1 are reasonable results achieved. This strategy seems to be premature in its reallocation of packages. The migration of a package to another depot does not take into account the vehicle assignments and the associated capacity of the vehicles. 1-Move on a feasible solution does take these constraints into account.

### 4.2.2 Order of Local Search Improvement Policies

Here we consider the sensitivity of solution quality to the ordering of local search improvement policies. In particular, we evaluate the effect of interchanging the order of 1-Move and 2-Swap in the initial stages of the winning policy program. The results are dramatic and demonstrate that the ordering of local search improvement policy applications can be significant.

Table 5 summarizes the effect of interchanging two improvement policies in the winning policy program. For all three example datasets, the results of the winning policy program cannot be achieved.

| (Value of Reported Results) | DataSet 1 (2263) | DataSet 2 (428.95) | Dataset 3 (689.85) |
|---|---|---|---|
| **Policy Program III** | | | |
| • Best-fit($R_{pd}$,distance-to-depot) | | | |
| • 1-Move($R_{pd}$,distance-to-nearest-neighbor) | | | |
| • Sort by Angle from Assigned Depot | | | |
| • Next-Fit($R_{pv}$,) | | | |
| • First-fit($R_{pr}$,) | | | |
| • Satisfy remaining relations | | | |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | 2341 | 486.8 | 870.2 |
| • 1-Move($R_{pr}$,distance) First-Fit($R_{pv}$,) | 2341 | 486.8 | 870.2 |
| • 2-Swap($R_{pr}$,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2276 | 481.3 | 785.8 |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | 2237 | 471.2 | 782.2 |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | 2178 | 469.2 | 777.6 |
| • 1-Move($R_{pr}$,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2175 | 461.4 | 777.4 |
| • 2-Swap($R_{pr}$,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2175 | 460.9 | 777.1 |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | 2175 | 459.4 | 774.9 |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | 2175 | 459.4 | 771.0 |

Table 4: A 1-Move Intermediate Local Search Program

| (Value of Reported Results) | DataSet 1 (2263) | DataSet 2 (428.95) | Dataset 3 (689.85) |
|---|---|---|---|
| **Policy Program IV** | | | |
| • First part of Policy Program I | 2532 | 464.8 | 712.4 |
| • 2-Swap($R_{pr}$,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2302 | 451.5 | 698.4 |
| • 1-Move($R_{pr}$,distance) First-Fit($R_{pv}$,) | 2302 | 445.2 | 697.1 |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | 2298 | 444.7 | 691.9 |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | 2292 | 428.0 | 682.5 |
| • 1-Move($R_{pr}$,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2290 | 428.0 | 679.5 |
| • 2-Swap($R_{pr}$,distance) First-Fit($R_{pv}$,) First-Fit($R_{pd}$,) | 2290 | 428.0 | 679.5 |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | 2290 | 428.0 | 679.5 |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | 2289 | 428.0 | 673.0 |

Table 5: Change in the Order of Improvement Policies

## 4.3 Policy Programs for a Highly Constrained Problem

Previous sections analyzed the effect of policy programs on relatively pure datasets. In each of the previously described datasets, all vehicles have the same capacity for packages, all depots have the same capacity for packages and the same capacity for vehicles, and all packages are identical in size.

This section investigates the performance of policy programs on a far more complicated dataset. This dataset portrays real world constraints more realistically. More specifically, we consider a DVRP problem in which depots have different capacity limits for vehicles and packages. Also, vehicles have different capacity limits for packages and are limited in the distance they can travel. Finally, packages have different sizes.

To construct such a dataset, we modified the first of the three datasets described in previous sections as follows. Package size is evenly distributed among the values 5,10,15,20,25,30,35. Vehicle capacity for packages is evenly distributed among the values 60,120,180. The depot with the highest utilization in previous solutions is limited to 300 package units and 3 vehicles. Other depots have capacity limits of 400 packages and 4 vehicles. Finally, each vehicle is limited to a travel distance of 400 units (where the package sites are in a plane of size 316 by 244 and travel time is equal to the euclidean distance). A close examination of the data will reveal the difficulty of this problem.

We constructed three policy programs to solve this problem. The first two programs attempt a solution by completing all assignments for one package before attempting the assignments for the next package. Each depot is considered, and within each depot all vehicles are considered and within the routes of the vehicle, the best position in the route is established.

These two policy programs differ in the order in which packages are placed. In policy program V, packages are sorted according to their angle to the closest depot. In policy program VI, packages are sorted according to their size.

| | Constrained DataSet |
|---|---|
| **Policy Program V** | |
| • Sort by angle to closest depot | |
| • Best-fit to closest depot | |
| Next-Fit($R_{pv}$,) | |
| First-fit($R_{pr}$,) | |
| Satisfy remaining relations | |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | Infeasible (4) |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| • 1-Move($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| First-Fit($R_{pd}$,) | |
| • 2-Swap($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| First-Fit($R_{pd}$,) | |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| • Sort leftovers by angle to closest depot | |
| • Best-fit leftovers to closest depot | |
| Next-Fit($R_{pv}$,) | |
| First-fit($R_{pr}$,) | |
| Satisfy remaining relations | Infeasible (3) |
| **Policy Program VI** | |
| • Sort by package size | |
| • Best-fit to closest depot | |
| Next-Fit($R_{pv}$,) | |
| First-fit($R_{pr}$,) | |
| Satisfy remaining relations | |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | Infeasible (12) |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| • 1-Move($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| First-Fit($R_{pd}$,) | |
| • 2-Swap($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| First-Fit($R_{pd}$,) | |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | |
| • 2-Reinsert($R_{pr}$,distance) First-Fit($R_{pv}$,) | |
| • Sort leftovers by size | |
| • Best-fit leftovers to closest depot | |
| Next-Fit($R_{pv}$,) | |
| First-fit($R_{pr}$,) | |
| Satisfy remaining relations | Infeasible (11) |

Table 6: Single Policy Solution

Policy programs V and VI were constructed to make two attempts to place an unconfigured package. If any packages remain unconfigured after the above improvement policies are applied (the number of unconfigured packages is listed inside the parentheses), a second attempt is made to place the packages. The basic idea is to "clean up" the partial solution in order to place more packages. That is, after the first attempt is made to place packages, a collection of improvement policies are applied to the partial solution. These improvement policies move and interchange packages between depots and vehicles if a decrease in total route distance results. After these policies are applied, an approximation improvement policy which establishes all assignments for a single package is again attempted.

Policy program VII uses interim local search as an integral part of the solution procedure. The initial assignment of packages to depots undergoes modification by a local search improvement policy. This program swaps the assignments of two packages if the total distance to the assigned depots is decreased. The assignment of packages to vehicles is performed next, and then packages in a vehicle are routed.

At this stage in policy program VII the relations which bind routes to vehicles and vehicles to depots are not satisfied. Two interim local search improvement policies are applied to facilitate a feasible assignment for these relations. The first simply applies 2-opt to each route. The second local search improvement policy moves a package from a route if that route is above the travel time limit specified by the route's vehicle (a route's vehicle can be determined by the transitivity) Packages in an infeasible route are moved to the route which results in the least increase in travel distance. Finally, policy program VII completes the remaining assignments of routes to vehicles and vehicles to routes are made.

As the results in Tables 6 and 7 indicate, only policy program VII achieves a feasible solution. The other two policy programs are unable to obtain a feasible solution because they do not adjust the initial assignments until all the resources are committed. If the results of this single example can be generalized, it would seem that the interim local search must be an integral part of the solution procedure on highly constrained datasets. It is not enough to place a collection of packages until no further progress can be made and then perform local search. Local search must accompany approximation procedures in each of the critical stages of a solution building.

Our analysis of the results from the two infeasible policy programs indicate that constraints can cause "log jams" and only a series of reassignments can correct the situation. These reassignments would need to be made in the

context of a local search policy program which exchanges configured objects with unconfigured objects.

| | Constrained DataSet |
|---|---|
| **Policy Program VII** | |
| • Best-fit($R_{pd}$,distance-to-depot) | |
| • 2-Swap($R_{pd}$,min-total-distance-to-depots) | |
| • Sort by Angle to Assigned Depot | |
| • Next-Fit($R_{pv}$,) | |
| • First-fit($R_{pr}$,) | |
| • 2-Opt($R_{pr}$,near-neighbors,distance) | |
| • 1-Move($R_{pr}$,adjacent-routes,route-feasibility) | |
| • Satisfy Remaining Relations | 2717 |

Table 7: Interim Local Search Solution

# 5 Conclusion and Directions for Future Work

In this paper we have presented a notation for solving complex hierarchically structured optimization problems. Optimization problems are specified using primitive assignment relationships that are linked together in a graph structure. This approach to problem decomposition has three immediate advantages: the reduction to basic primitives allows us to exploit well-behaved approximation algorithms in a natural way; the class of primitives admit well-understood local search algorithms which are generally efficient and effective; and the hierarchical structure of the notation allows us to satisfy multiple assignment relationships methodically and efficiently.

Building assignments that determine a problem solution is accomplished through a mechanism called an improvement policy. An improvement policy attempts to optimize or extend a given assignment or set of assignments using appropriate approximation and local search routines. Improvement policies are combined into policy programs using conditionals, looping constructs, and additional operators which perform extraction, sorting, etc.

We believe that by first specifying the HCG for a problem and then creating an appropriate and easily altered policy program, we have devised an effective new approach to solving complex optimization problems.

A prototype implementation of OPL has been written in Lisp. It has performed well on several examples including DVRP, which is reported on here. Solutions to the warehousing problem of section 2.5 and to a check-processing optimization problem are reported on in [Macleod, 89].

We are extending the research reported on here with work in several directions, which we enumerate below:

- **flexibility** How well does the OPL notation perform over a broad class of hierarchically structured optimization problems? Several issues bear on this assessment of adequacy. The OPL approach seems to exclude such traditional methods as dynamic programming and branch and bound techniques. Does this make the OPL approach too restrictive? (We note that simulated annealing and other randomization methods *are* consistent with the OPL approach.)

  Another flexibility issue concerns the extensibility of the notation. Are our current primitives, augmented with others such as, for example, approximation algorithms for 2-D rectangle layout, sufficient to represent a broad class of problems?

- **language issues** Our preliminary version of OPL has identified a

number of issues that must be addressed if OPL is to become a viable rapid prototyping language for complex optimization problems. These include: the role of transitivity of the HCG; the mechanism by which constraints are associated with objects and primitive assignment relationships; formal properties of primitive assignment relationships; a macro facility that allows primitives such as partitioning to be combined with other primitives such as slotted lists; and the ways in which the various effiencies described in section 3.2.1 can best be exploited.

- **generic environments for specific problem classes** OPL notation is applicable to a broad class of problems drawn from diverse domains in operations research and computer science. How can we specialize OPL in order to construct environments that apply to more specific problem classes? Currently we are developing requirements for several such generic environments, including vehicle routing, scheduling, and quadratic assignment problems.

- **rule-based local search** Many practical optimization problems include strange constraints and forbidden positions which make traditional application of local search difficult. How can a more flexible "rule-based" local search mechanism be formulated which will allow OPL to handle the unpredictable situations that frequently arise in real problems? For some initial work in this direction, see [Moll and Healy, 1988].

- **temporal issues** Many complex optimization problems involve temporal reasoning in an essential way. In particular this is true for scheduling problems. How can the HCG/policy approach we have developed be adapted to situations in which temporal considerations are important?

# 6 Bibliography

1. Beasley, J.E. (1983) Route-First-Cluster Second Methods for Vehicle Routing. OMEGA, The International Journal of Management Science, Vol 11, No. 4. Pgs 403-408.

2. Christofides, N. and J.E. Beasley, (1984) The Period Routing Problem. Networks, Vol 14, Pgs 237-256.

3. Fisher, M.L. and R. Jaikumar (1981) A Generalized Assignment Heuristic for Vehicle Routing. Networks, Vol 11, Pgs 109-124.

4. Garey, M.R. and Johnson D.S. (1979) Computers and Intractability: A guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco.

5. Gheysen, F. , B. Golden, A. Assad (1986) A New Heuristic for Determining Fleet Size and Composition. Mathematical Programming Study (26), Pgs 233-236.

6. Golden, B.L., C.C. Skiscim (1986) Using Simulated Annealing to Solve Routing and location Problems, Naval Research Logistics Quarterly, Vol. 33, Pgs 261-279.

7. Haimovich, M., A.H.G. Rinnooy Kan (1985) Bounds and Heuristics for Capacitated Routing Problems. Mathematics of Operations Research Vol 10, No. 4, Pgs 527-542.

8. Hillier, F.S. (1983) Heuristics: A Gamblers Roll. Interfaces 13 9-12.

9. Johnson, D.S., Papadimitriou, C.H: Yannakakis, M. (1985) How Easy is Local Search? Proceedings of the 25th Annual Symposium on Foundations of Computer Science, IEEE.

10. Kernighan, B.W. and Lin, S. (1973) An Effective Heuristic for the Traveling Salesman Problem Operations Research 21 498-516.

11. Kernighan and Lin (1970), An Effective Heuristic Procedure for Partitioning Graphs, BSTJ No.2

12. Kirpatrick, et. al. (1983) Optimization by Simulated Annealing, Science, May 1983.

13. Krone M.J. and K. Steiglitz (1974) Heuristic Programming Solution of a Flowshop-Scheduling Problem, Operations Research 22, no 3. Pgs 629-638.

14. Lin S., Computer solutions to the TSP, BSTJ No.10 Dec. 1965

15. MacLeod, B.B. (1989) OPL: A Notation and Solution Methodology for Hierarchically Structured Discrete Optimization Problems. Ph.D. Thesis, University of Massachusetts, Amherst, to appear.

16. Markland, R and Robert Nauss (1983) Improving Transit Check Clearing Operations at Maryland National Bank. Interfaces 13: February 1983.

17. Melhorn, K. (1984) Graph Algorithms and NP-Completeness Monographs on Theoretical Computer Science. Springer Verlag, New York.

18. Moll, R., and Healy, P. Towards Rule-Based Local Search, COINS Technical Report, 1988, to appear.

19. Nahar,S., Sahni S., Shragowitz, E. (1986) Simulated Annealing and Combinatorial Optimization 23rd Design Automation Conference. Pgs 293-299.

20. Papadimitriou C. and Steiglitz K. (1982) Combinatorial Optimization: Algorithms and Complexity. Prentice Hall, Englewood Cliffs, NJ.

21. Ratner D. and Pohl L. (1986) Joint and LPA*: Combination of Approximation and Local Search, Proceedings of AAAI-86.

22. Spaccamela, A.M., A.H.G. Rinnoy Kan, L. Stougie, (1984) Hierarchical Vehicle Routing Problems, Networks, Vol 14 Pgs 571-586

23. Steiglitz K., P. Weiner and D.J. Kleitman, (1969) The Design of Minimal Cost Networks, IEEE Transactions on Circuit Theory. CT-16, No. 4 Pgs 455-460.

24. Stewart, W.R., B.L. Golden (1984) A Lagrangean Heuristic for Vehicle Routing, European Journal of Operational Research (15), Pgs 84-88.

25. Tan, C.C.R and J.E. Beasley, (1984) A Heuristic Algorithm for the Vehicle Routing Problem. OMEGA, The International Journal of Management Science, Vol 12, No. 5. Pgs 497-504.