Optimizing Paint Blocking in an Automobile Assembly Line: An Application of Specialized TSP's

by

Joel Scott Sokol

B.S., Applied Sciences in Engineering
B.A., Mathematics and Computer Science
Rutgers University, 1994

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

Author .....
Department of Electrical Engineering and Computer Science
May 7, 1999

Certified by .....
Thomas L. Magnanti
Dean of Engineering and Institute Professor
Thesis Supervisor

Accepted by .....
James B. Orlin
E. Pennell Brooks Professor of Operations Research
Codirector, Operations Research Center

Optimizing Paint Blocking in an Automobile Assembly Line: An Application of Specialized TSP's

by

Joel Scott Sokol

Submitted to the Department of Electrical Engineering and Computer Science
on May 7, 1999, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

In the automobile manufacturing industry, vehicle production is an assembly-line process. Automobile companies typically sequence vehicle production based on workload balancing factors, with little consideration of vehicle colors. The resulting sequence usually has a small average paint block size. Because automobile manufacturers use expensive and sometimes pollutant chemicals to clean out old paint at each color change, they would like to increase the size of the paint blocks, while maintaining the original workload-balanced vehicle sequence. To achieve this goal, Ford and other automobile manufacturers are considering automated vehicle storage and retrieval systems that would allow them to perturb the original sequence around the vehicle painting station, creating larger paint blocks and then restoring the original sequence after painting. To evaluate these systems, it is necessary to develop a method for resequencing cars and for calculating the resulting savings in paint cleanings.

The problem of resequencing can be cast as a traveling salesman problem with time windows. For a real-life sequence size of 750 cars and windows of 75 slots per car in either direction, direct modeling using the strongest known TSPTW formulation yields an integer program with up to 200,000 constraints and 14,000,000 variables. Reduced formulations are still difficult to solve, even with the addition of polyhedral cuts.

We exploit special problem structure to solve the LP relaxation of this problem quickly using Lagrangean relaxation. We prove and use an *order-within-color* property to construct an enumerative formulation, and use a greedy approach to bound the LP optimum. We decompose the problem and solve smaller enumerative formulations sequentially to generate a heuristic solution that empirically is within 2.5% of optimality. Because our heuristic and bounding procedure runs in a total of one minute, an automobile manufacturer could use the process to adjust the resequencing process in real time to compensate for vehicles that have been delayed in the original sequence due to production defects or other disruptions. We also establish worst-case bounds ranging from 2.5 to 6 for another related heuristic.

Thesis Supervisor: Thomas L. Magnanti
Title: Dean of Engineering and Institute Professor

Acknowledgments

There are two important categories of people I would like to acknowledge: (i) those who helped me in one way or another with my work, and (ii) those who kept me sane by occasionally distracting me from my work.

Foremost in the first category is my advisor, Tom Magnanti. Tom always seems to know just what advice to give on any professional or research matter, and he never stops teaching – every time I visited his office, I learned something new. He was very supportive and flexible with regard to my schedule, encouraging me to pursue my outside interests even if it meant I was occasionally rushing out of research meetings with a musical instrument in each hand. If only he wasn't a Red Sox fan....

Jim Orlin and Dimitris Bertsimas posed some very interesting questions, both as teachers and as members of my thesis committee. Andreas Schulz also provided helpful insight and some useful references. Leon Hsu, Michael Vezza, and a legion of anonymous MIT computer consultants solved countless hardware and software problems for me. Lisa Bleheen always cheerfully helped me through scheduling, printing, and email compatibility difficulties, not to mention the trials and tribulations of PC/Macintosh interfaces. Last, but not least, the ORC administrative staff (Laura, Paulette, and Cheryl/Katana/Chris/Danielle) made things easier by shielding me from (and occasionally cutting right through) some of the Institute's red tape.

With all those people helping me work, it was important for me to have some people to help me *not* work. The ORC gang (Beril, Keely, Leon, Rebecca, and Tim), a horde of Next House residents (James, Jeff, Sarah, Tim, and the weekly poker players), and lots of musical groups (especially the MIT Marching Band) always gave me something fun to do. I got moral support (and visits) from my hometown friends (Dave, David & Angie, Dawn, and Leon & Jill) and my Rutgers friends (Jen & Mark, Pete & Casey, and Rob). The Hawthorne Caballeros Drum and Bugle Corps and the annual Pines Lake basketball club/Strat-O-Matic league (Brian, Dave, Joel, Sean, Steve, et al.) made sure that my summers also included some non-research activities. Elisabeth Pollio's friendship was omnipresent, by phone when not in person. Sarah Yohannan deserves special thanks for everything, including, at times, her patience.

My immediate family (my parents and my sister Sharon) and a number of other relatives (grandparents, aunts, uncles, and cousins) fall somewhere between the two categories and deserve their own separate thanks, especially for creating a family environment that encouraged me to pursue my education.

In all certainty ("with probability 1"), I have forgotten to acknowledge someone whose name should appear above. That person or persons have both my thanks and my apology.

One final acknowledgment is in order, to Chris Jakubowsky (one of my college roommates) for dragging me to my very first OR class eight years ago and unwittingly starting me down this path.

Contents

Chapter 1

Introduction and Literature Review

1.1 Problem Description

On an assembly line in the automobile industry, an automotive company will typically sequence cars based on a number of objectives, most dealing with line-balancing so that worker utilization is neither too high nor too low.

The three main stages of an assembly line are the body shop, the paint shop, and the trim and chassis installation. Cars flow through the assembly line from stage to stage in sequence (see Figure 1-1).

Figure 1-1: Stages of the automobile assembly process

In the first and last stages (the body shop and the trim, accessory, and chassis installation), different cars might require the installation of different sets of equipment; for example, one car might have a sun roof, and power windows, but no CD player, while another car might have a CD player and power windows, but no sun roof. To balance the workload, an automobile manufacturer will sequence cars so that even over small sets of consecutive cars, the frequency of each installation is approximately equal to its overall frequency. For example, if 10% of all cars have a sun roof, then exactly one out of every 10 consecutive cars in an ideal sequence would have a sun roof. If this were the case, a worker at a station installing sun roofs would have a fairly constant workload. Furthermore, by balancing workloads, the plant could avoid a backlog at the sun roof station that could slow down the line, as might occur if, for example, 10 consecutive cars required a sun roof. Of course, it is generally not possible to achieve this objective exactly for every characteristic, but sequencers attempt to get as close as possible to this desired property [4] [22].

Because the plant must paint every car, it faces no analogous "balance requirement" for the paint shop. Every car must pass through the paint shop and be painted by the same machinery; thus, the creation of paint blocks, consecutively-sequenced cars with the same required color at the painting station of the assembly line, is not a high priority. However, because the plant must clean the painting apparatus of old paint before painting a car of a new color, it can lose much time and money when the average paint block size is small. Atassi [4] estimates that an automotive plant incurs a cost of $15 or more each time it changes the paint color. Our data, taken from a single assembly line at a single automobile manufacturing plant, specifies that 750 cars per day will pass through the paint shop, and

each car will be painted one of 15 possible colors. If we were able to decrease the daily number of paint blocks by 500, we could save the automobile manufacturer on the order of $7500 each day, just on this single assembly line. This translates to a savings of millions of dollars per year, just for this one production line. Reducing the number of paint cleanings can also have a positive environmental impact, as the cleaning chemicals often contain environmental pollutants.

Automobile companies have considered three basic ideas to alleviate problem of small paint blocks while still retaining the original automobile sequences [13]. A fourth possibility is to not make any adjustments, and simply incur all of the paint-cleaning costs.

The first idea is to split the assembly line into two or more sub-lines before the paint shop (see Figure 1-2). The plant will then assign each car to one of the sub-lines. Whenever a new vehicle is to enter the paint shop, the plant can choose the entering car from among the next-in-line car at each sub-line (possibly a car of the same color as the last car painted). After exiting the paint shop, the cars are reshuffled via the same process to recreate the initial ordering [13]. This idea is a generalization of the current no-adjustment method (which has only one sub-line) and will decrease the paint-blocking costs. However, there is a cost tradeoff in that the sub-lines will be expensive to create and maintain.

A second possibility is to split the assembly line into two or more sub-lines, like the previous method. However, at any time that a car is next-in-line in one of the sub-lines, it can be shifted to the end of its sub-line instead of being sent into the paint shop [13] (see Figure 1-3). Again, there is a tradeoff between the cost of building and maintaining the sub-lines and the savings gained through better paint-blocking.

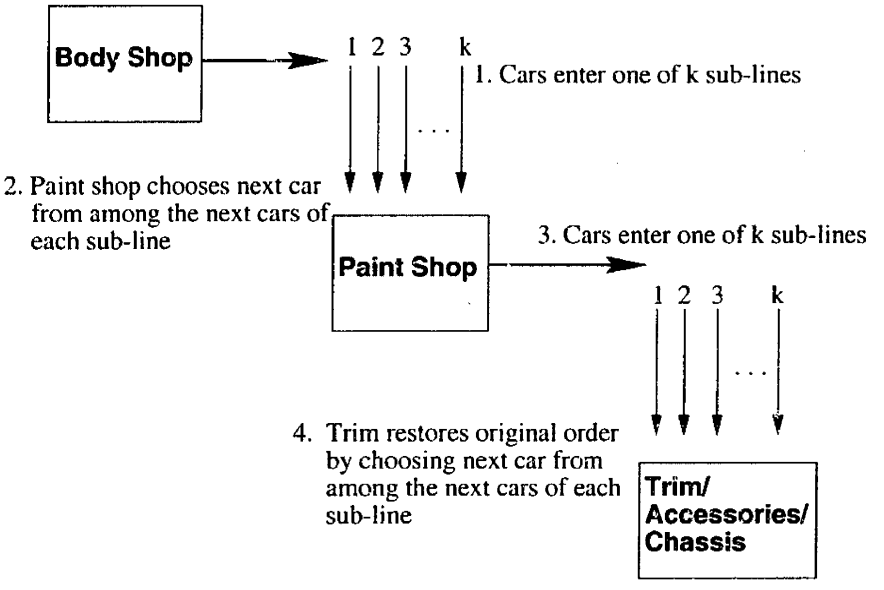

Figure 1-2: k-sub-line example with no balking

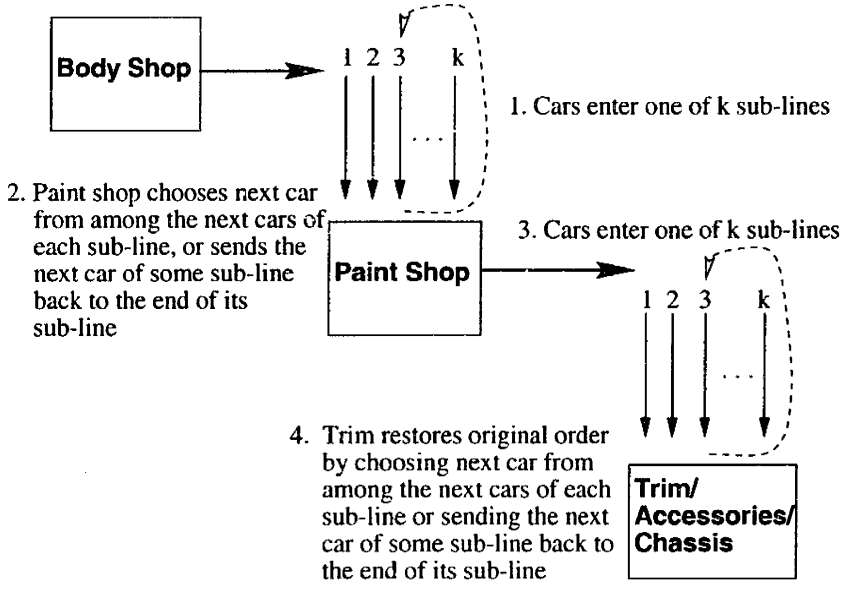

Figure 1-3: k-sub-line example with balking

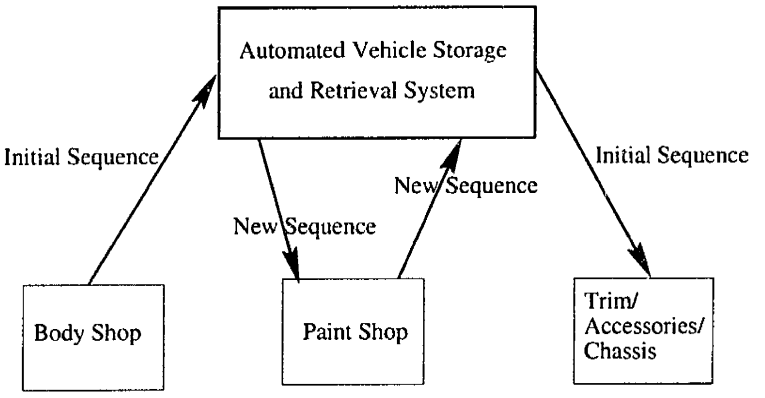

Figure 1-4: Resequencing with automated vehicle storage and retrieval system

As a third option for better paint-blocking, Atassi [4] and Myron [22] have proposed the use of temporary resequencing, facilitated by an automated storage system (see Figure 1-4). The automated storage system would act as a buffer that can store cars before and/or after painting. Using this buffer, a plant can perturb the order for painting cars to create larger paint blocks, and then restore the original sequence after painting.

To decide which of the four alternatives is best for a certain assembly line, it is important to know the cost savings generated by each type of line. Because some of the associated optimization problems are difficult, the cost savings will be related to the best available heuristic solution, rather than the optimal solution. Moreover, because the solutions must be available in real time, the speed of generating the heuristic solutions is also important.

In this thesis, we study the fourth option, using the automated vehicle storage and retrieval system, comparing it to the "baseline" solution of not adding any extra equipment.

A simple example with two gray cars and one white car illustrates the use of an automated storage system (see Figures 1-5 through 1-11). Instead of painting the white car between the two gray cars, we would like to paint the two gray cars first, and then the white car. We

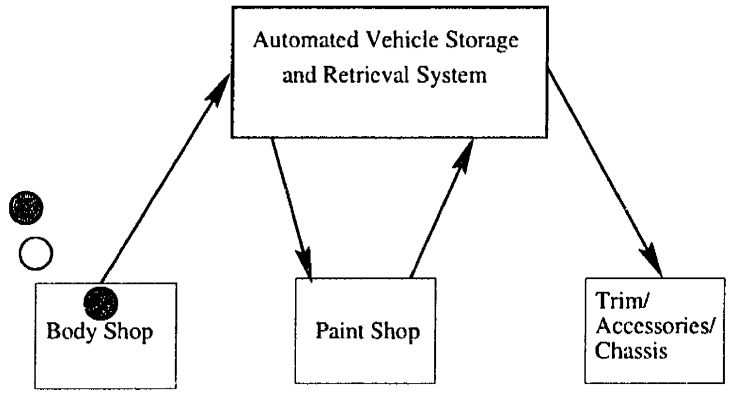

Figure 1-5: Step 1: First car enters body shop, other cars wait in order

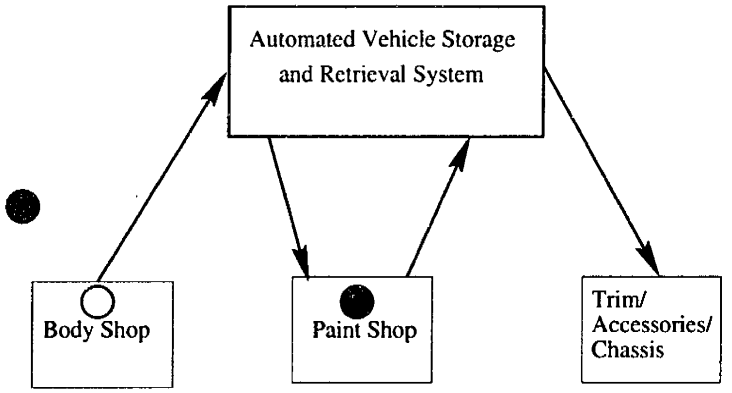

Figure 1-6: Step 2: First car advances to paint shop, second car enters body shop

can paint the first gray car and send it down the assembly line, move the white car into the storage system, and paint the second gray car. We can then move the second gray car into the storage system, paint the white car and send it down the line, and then send the second gray car down the line. In this way, we have reduced the number of paint changes from two to one while retaining the prespecified sequence of cars at every other stage of the assembly line.

The optimization problem induced by this situation is that of minimizing the total number of paint-cleanings (or, equivalently, maximizing the size of the average paint block) given the initial ordering of automobiles, the colors they are to be painted, and the maximum

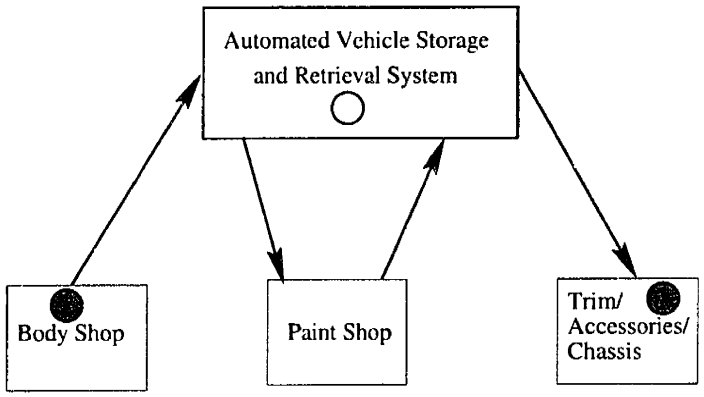

Figure 1-7: Step 3: First car advances to trim, second car waits in storage, third car enters body shop

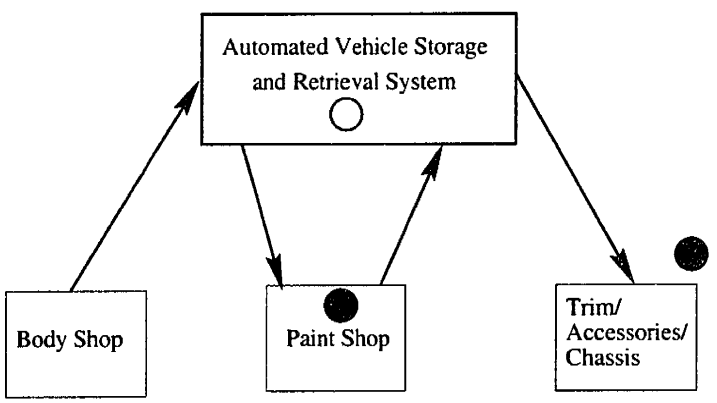

Figure 1-8: Step 4: First car finishes, second car waits in storage, third car advances to paint shop

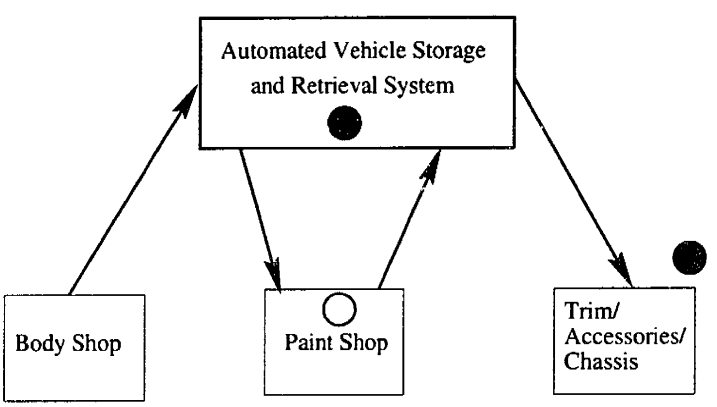

Figure 1-9: Step 5: Second car advances to paint shop, third car waits in storage

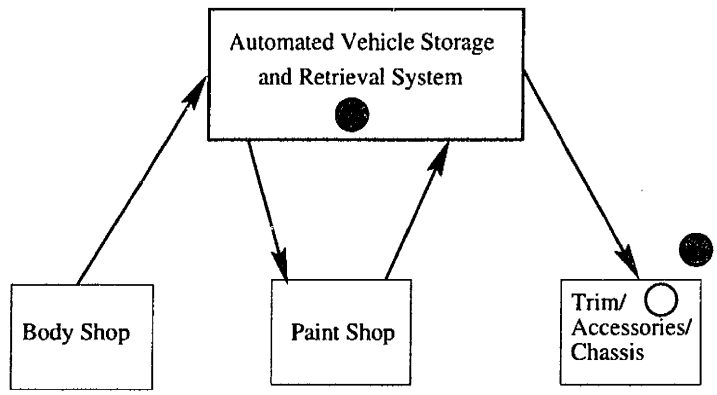

Figure 1-10: Step 6: Second car advances to trim, third car waits in storage

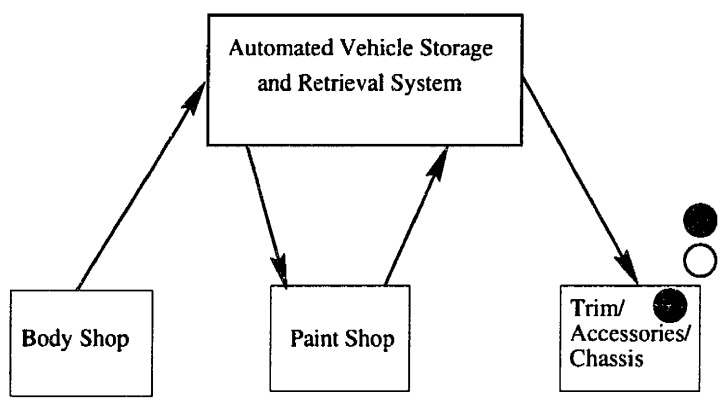

Figure 1-11: Step 7: Second car finishes, third car advances to trim

allowable perturbation in the sequence (dependent on the size of the buffer).

1.2 Relation to the Traveling Salesman Problem

The traveling salesman problem (TSP) is traditionally defined by a set of cities and a set of distances between pairs of cities. The problem is to determine the shortest-distance route (tour) that visits every city exactly once and then returns to the first city visited. The TSP has a rich and storied history, and is known to be an NP-hard problem [19].

A variant of the TSP, called the TSP with time windows (TSPTW), includes an extra set of constraints, specifying the earliest and latest times that the tour can visit each city. For example, a city's "time window" constraint might specify that the tour must visit the city between three and six days after the route begins. Because a regular TSP is a TSPTW with large time windows, the TSPTW is also an NP-hard problem [18]. In fact, Savelsbergh [27] has proved that the problem of finding any feasible solution to the TSPTW is NP-hard, even when travel is permitted between any pair of cities (a case where finding a feasible solution to the standard TSP is easy).

Our problem is a TSPTW. Cars correspond to cities that must each be painted once, in a certain order. We need to paint each car within a certain time window. Specifically, if we define w to be the maximum allowable perturbation in the sequence, then the ith car in the original sequence must be between the $(i - w)$th and $(i + w)$th car (inclusive) in the perturbed sequence. We define the "distance" between cars to be either one (if they are different colors and necessitate the creation of a new paint block) or zero (if they are the same color and do not necessitate the creation of a new paint block).

Researchers have used the TSPTW to model a number of other real-world problems, including buoy tender routing and scheduling [9] and dairy routing [1]. In the general case, as treated by most of the TSPTW literature, the problem models a single-vehicle routing problem with service windows.

Various researchers, including Ascheuer, Fischetti, and Grotschel [3], Balas and Schulz [5], and van Eijl [31], have studied the polyhedral structure of the TSPTW. However, finding the dimension of the TSPTW is NP-hard [5], so determining whether inequalities define facets of the polytope is also a hard problem.

Researchers have proposed a number of exact algorithms for the TSPTW. Tsitsiklis [29] studied special cases of the TSPTW, and proposed an algorithm for a generalized version of the problem with a small number of locations, some needing to be visited more than once. We will apply a similar strategy in this thesis to demonstrate that our problem is polynomially solvable; however, the polynomial is of degree 15 for real-world-sized problems, and thus the solution method is unsuitable for our problem.

Dumas, Desrosiers, Gelinas, and Solomon [12] improved upon the generic dynamic programming algorithm by imposing partial orderings on the cities based on the time constraints. We will incorporate similar ideas in our model when we prove and exploit the order-within-color property.

Pesant, Gendreau, Potvin, and Rousseau [26] proposed a constraint logic programming algorithm for solving the TSPTW exactly. They achieved good results on some general problems in the literature (up to 60 cities with a window size of 20), but the solution methodology requires too much time for a problem of our size (750 cities and a window size

of 75). The cutting-plane approach suggested by Malandraki and Daskin [20] encountered the same difficulties, while achieving slightly worse results for smaller problems.

Simonetti and Balas [11] proposed a linear-time algorithm for solving the TSPTW; however, the linear-time algorithm involves exponential memory usage.

Other researchers have applied generalized heuristics, such as tabu search (Carlton and Barnes) [7] and genetic algorithms (Nygard and Yang) [24], with some success to the TSPTW, but the algorithms are able to solve only much smaller problems than what we encounter in our application. Problem-specific heuristic methods, such as decomposition (Mukund) [21], insertion (Gendrau, Hertz, Laporte, and Stan) [14], nearest-neighbor (Malandraki and Daskin) [20], and interchange (Savlesbergh; Vander Weil and Sahinidis) [27] [32], solve general problems in a matter of minutes to within 4-10% of optimality, but are similarly unwieldy for the sequence length and window size of our problem.

Our particular TSPTW has a specially structured objective function: every distance is either zero or one. Since we must incur the cost of exactly the same number of inter-car distances ($N - 1$, when the sequence contains N cars) in any feasible solution, we could create an equivalent problem by redefining the distances as one and two (instead of zero and one).

Papadimitriou and Yannakakis [25] have studied the special case of the symmetric TSP with all distances equal to one or two (this problem is still NP-hard), and have constructed a polynomial-time $\frac{7}{6}$-approximation algorithm for the problem. The algorithm relies heavily on the fact that any permutation of the cities defines a feasible solution. The TSPTW does not satisfy this property, so Papadimitriou and Yannakakis' algorithm is not applicable to

our problem.

Overall, the "state of the art" TSPTW algorithms from the literature are able to solve problems of up to 200 cities and a window size of 40, with solution times for the largest cases measured in hours. As mentioned previously, the real-world data motivating our study has a sequence of 750 cars. The data also specifies a half-window size w of 75 cars (75 cars in either direction, for a total window size of $2w + 1 = 151$) [4].

The data set contains a total of 15 colors, and a probability distribution on the frequency of each color's occurrence [4]. Later, we will discuss other cases with different sizes of the color set, window, and automobile sequence.

Chapter 2

Models

In this chapter, we present a series of models for the problem. As we describe later, some of these models are too large to solve directly when applied to the full-scale real world data. Therefore, we use a smaller test problem to obtain a general idea of the performance of our formulations. We generate random examples having 200 cars, 15 colors (with the same color distribution as in the full-scale example), and a half-window size of 4 to use as test problems. The results we report for our test problem are for a representative instance with an initial sequence containing 184 paint blocks and an optimal solution containing 114 paint blocks, and so 86 transitions between cars of the same color. In our optimization models, we choose to maximize the number of transitions between cars of the same color rather than (equivalently) minimizing the number of paint blocks.

2.1 Formulations with Integer Polyhedra

2.1.1 Quadratic Assignment Formulation

One natural formulation would be to model the problem as an assignment problem, with N cars to assign to N slots in the painting order. In this formulation, $x_{ij} = 1$ if and only if we place car i in slot j.

$$
\begin{aligned}
[QAF:] \text{ maximize } & \sum_{j=1}^{N} \sum_{\substack{c_i = c_k \\ \max(N, i+w)}} x_{ij} x_{k,(j+1)} & & & \\
\text{subject to } & \sum_{\substack{j=\min(1,i-w) \\ \max(N,i+w)}} x_{ij} & = & 1 & \forall i \\
& \sum_{i=\min(1,i-w)} x_{ij} & = & 1 & \forall j \\
& x_{ij} & \in & \{0,1\} & \forall (i,j).
\end{aligned}
$$

The standard assignment constraints restrict each car to exactly one slot, and each slot to exactly one car. The objective counts the number of pairs of consecutive cars of the same color.

The assignment problem polyhedron is integral; however, the objective function is neither convex nor concave, so this model is difficult to solve, even if we relax the integrality constraints.

We can exploit the 0/1 structure of the objective function to create a concave objective function. If we add $-\frac{N^2}{2} \sum_i \sum_j x_{ij}^2$ to the objective function, it becomes the sum of $-\frac{1}{2}(x_{ij} - x_{k,j+1})$ terms and $-\frac{1}{2} x_{ij}$ terms, all of which are concave. Since exactly N of the x_{ij} will be one in any feasible solution, adding this term to the objective subtracts N^3 from the objective value of every feasible solution, and thus does not alter the optimal solution.

When we relax the integrality constraints for this new problem, we are maximizing a nonlinear, concave cost function, so this is an easy problem to solve. However, the optimal solution will generally not lie at one of the extreme points, so the solution generated will not correspond to an assignment of cars to slots. Alternatively, we could transform the objective function to be the maximization of a convex function (by adding $\frac{N^2}{2} \sum_i \sum_j x_{ij}^2$ to the objective function), in which case the optimal solution would lie at an extreme point; however, this new problem is still hard to solve.

2.1.2 Linearization

To remove the difficult nonlinearities, we could linearize the problem (see Figure 2-1) by replacing the objective function by the sum of its partial derivatives evaluated at any feasible point. If we solve the linearization of problem QAF, we do obtain an extreme point solution; however, it contains 161 paint blocks, and we show later that the optimal solution is 114 paint blocks, so the solution of the linearized linear program is 41% from the optimal solution to the integer program. We can understand why this method might perform poorly by considering the structure of the linearized problem. The constraints were initially linear, so they do not change. The linearized objective function simply counts the number of cars in a feasible solution, or N. Thus any feasible solution will have the same objective value, so there is no differentiation between feasible solutions.

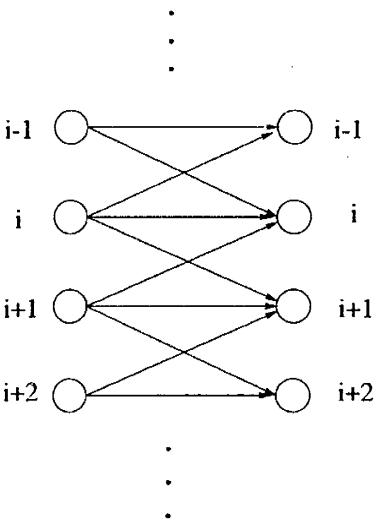

Figure 2-1: Network Representation of Linearized Quadratic Assignment Formulation with $w = 1$

2.1.3 Sequence Assignment Formulation

Another possible formulation with an underlying integral polyhedron uses only variables z_{ij} (see Figure 2-2), defined to be $z_{ij} = 1$ if and only if car j is assigned to the slot immediately after the slot to which car i is assigned. We define $R(i)$ to be the set of cars that could possibly be in the slot after the slot assigned to car i. In general, $R(i)$ contains all cars between $\max(1, i - 2w + 1)$ and $\min(N, i + 2w + 1)$, inclusive.

$$
\begin{aligned}
[SAF:] \text{ maximize } & \sum_{c_i = c_j, j \in R(i)} z_{ij} & & & \\
\text{subject to } & \sum_{j \in R(i)} z_{ij} & = & 1 & \forall i \\
& \sum_{i: j \in R(i)} z_{ij} & = & 1 & \forall j \\
& z_{ij} & \in & \{0,1\} & \forall i, j.
\end{aligned}
$$

The first set of constraints states that all cars have one successor, and the second set of

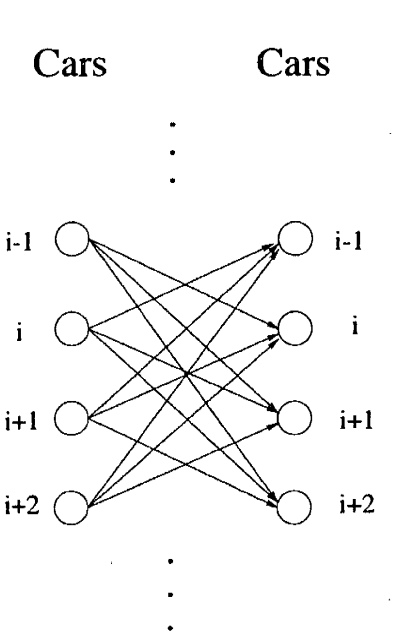

Figure 2-2: Network Representation of Sequencing Assignment Formulation with $w = 1$

constraints states that all cars have one predecessor. We add a dummy car at the beginning and at the end of the sequence so that the constraints will hold for the first and last cars.

Unfortunately, this formulation need not generate solutions that are feasible for the original problem. For example, consider the simple instance of Figure 2-3 with $N = 5$ cars, $C = 2$ colors, and a window of $w = 1$. The mapping $(3, 2, 1, 4, 5)$ of cars to slots (car 3 in slot 1, car 2 in slot 2, etc.) is a feasible solution to this formulation, but would not be a feasible solution to the original problem, because cars 1 and 3 have each moved 2 slots (greater than the window size of 1) from their placement in the initial sequence. The solution cost of 1 is lower than the optimal solution cost of 2.

However, since no feasible solution to our problem is infeasible in this formulation, we can use the formulation to provide an upper bound on the optimal solution. For our test problem instance, the optimal objective value of this relaxation is only 141, 64% more than the objective value of an optimal solution.

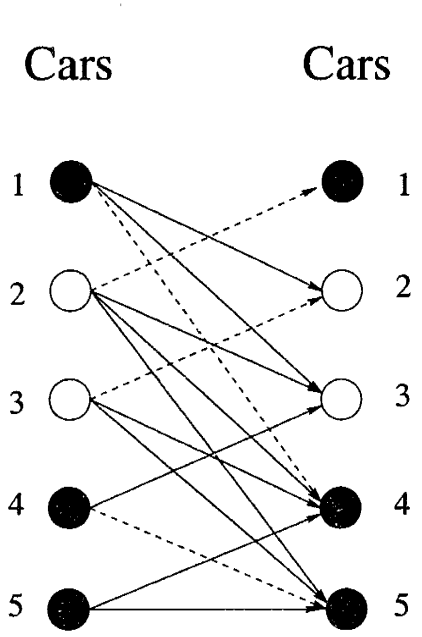

Figure 2-3: Feasible SAF Solution (Infeasible for Original Problem)

2.1.4 Conclusions

While an exact linear formulation with a simple integral polyhedron (such as an assignment problem polyhedron) would be ideal, it seems unlikely that one exists, given the problem's similarity to the traveling salesman problem. Thus, we attempt to find other formulations that might give better results than the simpler formulations.

2.2 Formulations With Non-Integral Polyhedra

If we allow ourselves to use formulations with nonintegral polyhedra, we can construct alternative formulations.

The difficulty with using only assignment constraints in a formulation of the problem is that the objective function is difficult to model using only assignment variables. By adding extra variables and/or adding a third dimension of indices to the core assignment model, we

improve the quality of our bounds.

2.2.1 Adding Car-Sequencing Variables

One way of avoiding the problems with multiplying variables in the objective function is to define a new set of variables. These new variables y_{ik} will specify whether car k immediately follows car i in the new sequence. We refer to this formulation as the Sequencing Variable Formulation, or SVF.

$$
\begin{aligned}
[SVF:] \text{ maximize } & \sum_{c_i=c_k} y_{ik} & & & \\
\text{subject to } & \sum_{j=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall i \\
& \sum_{i=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall j \\
& x_{ij} + x_{k,(j+1)} & \geq & 2y_{ik} & \forall i,j,k \\
& x_{ij} & \in & \{0,1\} & \forall i,j \\
& 0 \leq \; y_{ik} & \leq & 1 & \forall i,k.
\end{aligned}
$$

In this formulation, the first two sets of constraints are assignment constraints. The third set of constraints forces y_{ik} to be 0 unless cars i and k are assigned to consecutive slots. For all other pairs of cars, the optimization will set y_{ik} to value one since we are maximizing.

Unfortunately, the linear programming relaxation of this formulation is extremely weak. For our example problem with an optimal objective value of 86, the relaxation of SVF has an optimal objective value of 199.

When we applied branch-and-bound to solve this formulation for our simple test problem, we were unable to solve the problem, or even decrease the upper bound below the value 190,

within 6 hours of computation time.

2.2.2 Assignment-Based Formulation With Savings Indicator Variables

As another modeling approach, we could use binary indicator variables I_{ij} in the formulation, with variable I_{ij} equaling 1 if car i is in slot j and the next car of the same color is in slot $j+1$. The indicator variable I_{ij} specifies whether we obtain a cost savings when we assign car i to slot j, depending upon the car assigned to slot $j+1$.

We refer to this formulation as the Savings Indicator Formulation, or SIF.

$$
\begin{aligned}
[SIF:] \text{ maximize} \quad & \sum_{i,j} I_{ij} & & & \\
\text{subject to} \quad & \sum_{j=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall i \\
& \sum_{i=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall j \\
& 2I_{ij} - x_{ij} - \textstyle\sum_{k:c_i=c_k} x_{k,j+1} & \leq & 0 & \forall i,j \\
& x_{ij} & \in & \{0,1\} & \forall i,j \\
& I_{ij} & \in & \{0,1\} & \forall i,j.
\end{aligned}
$$

The third set of constraints restrict the indicator variable I_{ij} to be zero unless the ith car is in slot j and another car of the same color is in slot $j+1$.

This formulation is larger than the previous assignment-based formulations, and does not achieve better results; the relaxation also has an optimal objective value of 199.

2.2.3 Disaggregated Formulation With Savings Indicator Variables.

The Savings Indicator Formulation is weak because of the third constraint, $2I_{ij} - x_{ij} - x_{k,j+1} \leq 0$. If we set only one of the two x-variables to value one, the linear program can set the indicator variable to $\frac{1}{2}$. However, if we disaggregate the constraint, we eliminate this possibility and tighten the formulation. We refer to the tighter formulation as the Disaggregated Savings Indicator Formulation, or $DSIF$.

$$
\begin{aligned}
[DSIF:] \text{ maximize} \quad & \sum_{i,j} I_{ij} & & & \\
\text{subject to} \quad & \sum_{j=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall i \\
& \sum_{i=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall j \\
& I_{ij} - x_{ij} & \leq & 0 & \forall i,j \\
& I_{ij} - \textstyle\sum_{k:c_i=c_k} x_{k,j+1} & \leq & 0 & \forall i,j \\
& x_{ij} & \in & \{0,1\} & \forall i,j \\
& I_{ij} & \in & \{0,1\} & \forall i,j.
\end{aligned}
$$

The third and fourth sets of constraints restrict the indicator variable I_{ij} to be zero unless the ith car is in slot j and another car of the same color is in slot $j+1$. These two sets of constraints are a disaggregation of the sequencing constraint of the formulation SIF.

This formulation is larger than the previous formulation, but has a stronger linear programming relaxation; the upper bound for the small test problem is 152.6.

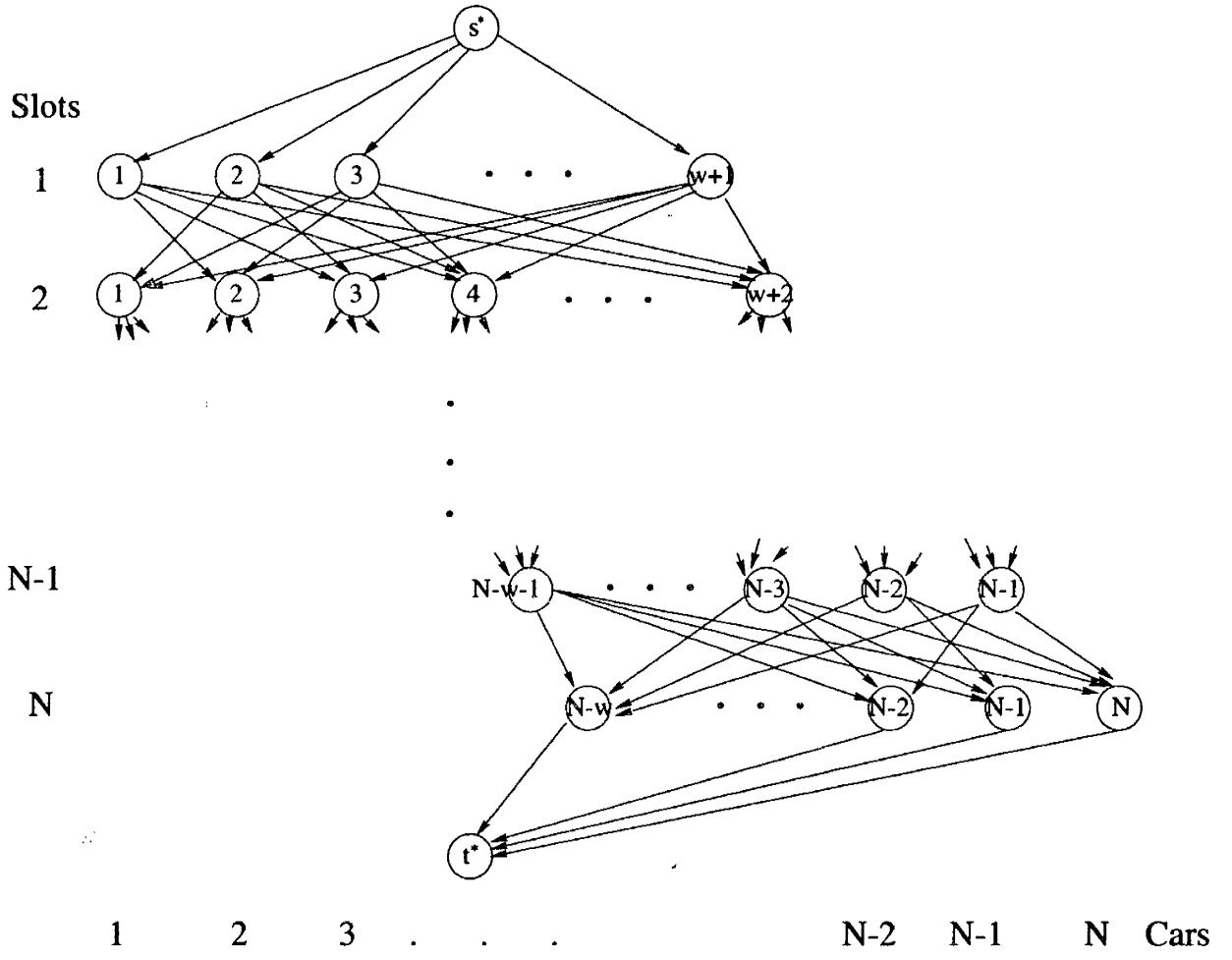

Figure 2-4: Network Representation of the $SP3$ Formulation

2.2.4 3-Dimensional Shortest-Path-Based Formulation

To avoid the problems that arise when we attempt to use a nonlinear formulation, we can add a third index to our variables. Let $x_{ijt} = 1$ if and only if car i is in slot t and car j is in slot $t+1$. Let $S(t)$ be the set of cars that can be assigned slot t in a feasible solution, and $\hat{S}(t) = \{(i,j) : i \in S(t), j \in S(t+1)\}$.

If we create dummy source and sink nodes s^* and t^*, we can think of this formulation as being based on the shortest path problem (see Figure 2-4). We have one node for each

possible car/slot assignment, and an arc between each possible pair of consecutive car/slot assignments. The source node has arcs to each of the possible assignments for slot 1, and the sink node has arcs from each of the possible assignments for slot N. If we define the costs on the arcs to be 0 if the colors of the endpoint cars are the same and 1 if the colors are different, the objective is then to find the shortest path through this network, subject to the constraint that each car must be used once. Thus, we refer to this formulation as the 3-Dimensional Shortest-Path-Based Formulation, or $SP3$.

We make one minor modification to obtain our final $SP3$ formulation; instead of the cost structure described above, we define the "costs" to be savings. A savings of 1 is assigned to each arc with same-colored endpoint cars, and a savings of 0 is assigned to each arc with different-colored endpoint cars. The objective then becomes finding the longest path. Since the underlying network is acyclic, doing so is no more or less difficult than finding the shortest path, still subject to the constraint that we must use each car exactly once.

$$
\begin{aligned}
[SP3:]\ \text{maximize} \quad & \sum_{c_i=c_j, i\in S(t), j\in S(t+1)} x_{ijt} & & & \\
\text{subject to} \quad & \sum_{(i,t):(i,j)\in\hat{S}(t)} x_{ijt} & = & 1 & \forall j \\
& \sum_{j\in S(t+1)} x_{ijt} - \sum_{k\in S(t-1)} x_{kit} & = & 0 & \forall (i,t) : i \in S(t) \\
& \sum_{j\in S(1)} x_{s^* j1} & = & 1 & \\
& \sum_{i\in S(N)} x_{it^* N} & = & 1 & \\
& x_{ijt} & \in & \{0,1\} & \forall (i,j,t) : (i,j) \in \hat{S}(t).
\end{aligned}
$$

The first set of constraints ensures that each car will be used once. The second set of constraints are flow balance constraints in the network of cars shown in Figure 2-4. The third and fourth constraints are flow balance constraints at the dummy source and dummy sink.

Formulation SAF can generate infeasible solutions, and formulation QAF has nonlinearities in the objective function. This formulation ($SP3$) avoids both problems, at the expense of an order of magnitude increase in the size of the formulation. Formulations QAF, SAF, SVF, and $SIF/DSIF$ each had $O(N^2)$ variables and $O(N)$ or $O(N^2)$ constraints. Formulation $SP3$, on the other hand, has $O(N^3)$ variables and $O(N^2)$ constraints.

Assuming that $W < \frac{N}{2}$, the time-indexed formulation has one flow-balance constraint for each feasible car/slot assignment. Thus it has a total of $2NW + 2N - 2W^2 + 4W - 2$ constraints. Since it contains one variable for each feasible pair of consecutive car/slot assignments, it has a total of $4N/W^2 + 2NW + N - \frac{10}{3}W^3 - 5W^2 - \frac{5}{3}W + 1$ variables. Obviously, the number of variables is much greater than the number of constraints.

This formulation is equivalent to a special case of Picard and Queyranne's [15] formulation for the time-dependent TSP, a modification of the TSP in which distances between cities are time-dependent. In our case, the distances are either zero or one (if the sequence is feasible for the specific pair of consecutive slots) or infinity (if infeasible). The formulation is also a special case of Vander Weil and Sahinidis' three-index formulation for the general TSPTW [32]. Gouveia and Voss [3] have shown that this formulation is at least as strong a formulation as any of the other standard formulations, with the exception of the multicommodity flow and subtour elimination formulations, for which no relationship is

known.

For our test instance, the relaxation of TIF gives an upper bound of 119.247, a marked improvement over any of the previously-discussed formulations, but still 39% short of the optimal solution (86). CPLEX branch-and-bound was unable to solve the problem or decrease the upper bound significantly within two hours.

Adding Subtour Elimination Constraints

We can improve the shortest-path-based formulation by adding the equivalent of subtour-elimination constraints. In a standard TSP formulation, these constraints $\sum_{i,j \in R} x_{ij} \leq |R|-1$ restrict any proper subset R of the nodes from being joined by more than $|R|-1$ edges. The analogous constraint in our formulation $SP3$ is $\sum_{i,j,t:i,j \in R, i \in S(t), j \in S(t+1)} x_{ijt} \leq |R|$.

There are $O(2^N)$ possible subtour breaking constraints, so we did not want to add all of them. Instead, we first added only the degree-2 subtour breaking constraints; that is, the constraints corresponding to cardinality-2 subsets of the cars. The addition of these constraints decreased the upper bound on our test instance to 104.806, or 22% from optimality. However, this was still not enough for CPLEX branch-and-bound to significantly decrease the upper bound or solve the problem.

We added degree-3 subtour breaking constraints to the formulation, which increased the generation time for the linear program file by an order of magnitude, tripled the linear programming solution time in CPLEX, and decreased the upper bound only to 101.929. Adding the degree-4 subtour breaking constraints was even less fruitful, producing a decrease to only 101.870. In both cases, CPLEX was still unable to solve the problem or decrease the

upper bound significantly.

2.3 Order-Within-Color Property

Our objective is to minimize the number of paint blocks: we gain a savings of 0 for each pair of consecutive cars that are to be painted different colors, and a savings of 1 for each pair of cars that are to be painted the same color.

For this cost structure, we can prove an order-within-color property. Given any feasible reordering of the cars, it is possible to construct another feasible reordering of the same cost with the property that cars of the same color appear in the reordering in the same sequence that they appeared in the original ordering. Although we state some of our later results based on this property only for the specialized zero/one cost structure, the results are valid for a more generalized cost structure with cost R_{kl} for painting color k immediately after color l.

Consider any feasible reordering P of the K cars. Let p_k be the slot of car k in the reordering. Thus, for every car k, $\min(1, k-w) \leq p_k \leq \max(K, k+w)$. Suppose that $p_j < p_i$ for two cars i and j, $i < j$, of the same color (that is, $c_i = c_j$). So, in the initial sequence, car i came before car j, but in the final sequence, car j is slotted before car i.

Now consider a similar reordering $\hat{P}$ of the original sequence, with $\hat{p}_k = p_k, \forall k \neq i, k \neq j$. Every car except for i and j are in the same slot in $\hat{P}$ as in P. Cars i and j swap places in the new reordering, so that $\hat{p}_i = p_j$ and $\hat{p}_j = p_i$.

Figure 2-5 shows an example of a feasible perturbed sequence and its corresponding feasible order-within-color solution.

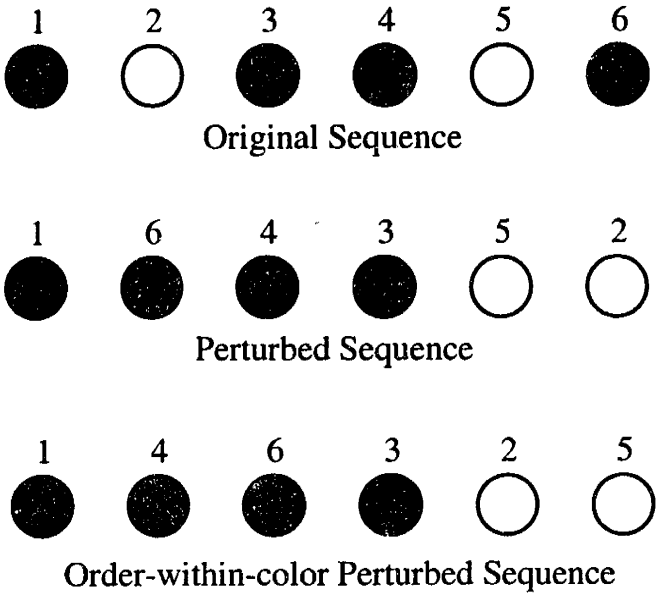

Figure 2-5: Illustration of perturbation and the order-within-color property

Since P is a feasible reordering, every car except i and j must be in an allowable slot in $\hat{P}$. Also, $\hat{p}_i = p_j < p_i \leq \max(N, i+w)$ and $\hat{p}_i = p_j \geq \min(1, j-w) \geq \min(1, i-w)$ (since $i < j$), so $\hat{p}_i$ is an allowable slot for car i. Similarly, $\hat{p}_j = p_i > p_j \geq \min(1, j+w)$ and $\hat{p}_j = p_i \leq \max(N, i-w) \leq \min(1, j-w)$, so $\hat{p}_j$ is an allowable slot for car j. Thus every car is in an allowable slot, so $\hat{P}$ is also a feasible reordering.

Notice that, because $c_i = c_j$ (cars i and j are the same color), the number of paint blocks in reordering P is the same as the number of paint blocks in reordering $\hat{P}$.

Thus, we will never have any incentive to switch the order relative to each other of two cars of the same color. That is, given two cars i and j, $i < j$, with $c_i = c_j$, we will never have an incentive to create a reordering Q with $q_j < q_i$.

Therefore, given any such reordering, we can always create an equivalent cost ordering with all cars ordered within their colors.

Order-within-color Property For any feasible reordering Q, a feasible reordering $\hat{Q}$ with $\hat{q}_i < \hat{q}_j$ whenever $i < j$ and $c_i = c_j$ contains exactly the same number of paint blocks as Q.

This result also applies for the more general case, when costs are dependent only on the colors of the cars being painted, but not on the specific cars or slots. For example, the plant might incur a cost $c_{blue,green}$ for painting blue and green cars in any two consecutive slots.

Because any feasible solution has an equivalent solution satisfying the order-within-color property, we restrict our feasible set to order-within-color solutions for the remainder of this work.

2.3.1 Effect of Order-Within-Color

In observing the fractional solutions generated by the $DSIF$ formulation and the $SP3$ formulation with and without subtour breaking constraints, we noticed that the solutions contained many instances of variables with a savings of one that would violate the order-within-color property. Since we have shown that restricting the set of feasible solutions to those that obeyed the order-within-color property would not change the optimal objective value, we can eliminate the violating solutions by adding the order-within-color restriction to our formulations.

Rather than add constraints to the formulations, we can enforce the restriction by changing coefficients and eliminating variables. For each car i, we define cnext(i) to be the next car of the same color as car i. In the $DSIF$ formulation, we modify the coefficients of each

constraint $I_{ij} - \sum_{k:i,k \text{ same color}} x_{k,j+1} \leq 0$. For every car $k \neq \text{cnext}(i)$, we set the coefficient of $x_{k,j+1}$ to be zero in the constraint. In this way, we eliminate the bonus for sequences that do not obey the order-within-color property, resulting in the following order-within-color disaggregated savings indicator variable ($ODSIF$) formulation:

$$
\begin{aligned}
[ODSIF:] \text{ maximize } & \sum_{i,j} I_{ij} & & & \\
\text{subject to } & \sum_{j=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall i \\
& \sum_{i=\min(1,i-w)}^{\max(N,i+w)} x_{ij} & = & 1 & \forall j \\
& I_{ij} - x_{ij} & \leq & 0 & \forall i,j \\
& I_{ij} - x_{\text{cnext}(i),j+1} & \leq & 0 & \forall i,j \\
& x_{ij} & \in & \{0,1\} & \forall i,j \\
& I_{ij} & \in & \{0,1\} & \forall i,j.
\end{aligned}
$$

We enforce order-within-color more directly in the $SP3$ formulation by simply eliminating all variables that correspond to cars of the same color, but not sequential within their color, being assigned to consecutive slots. For example, if cars i, j, and $k, i < j < k$, were all the same color, we would remove variable x_{ikt} for all relevant slots t. We could also remove all variables x_{ijt} for cars i and j of the same color and $j < i$. These changes, which are equivalent to setting the eliminated variables to zero, have the added benefit of reducing the problem size. We refer to this final three-dimensional formulation ($SP3$ plus the order-within-color restrictions, but without subtour breaking constraints) as $OSP3$.

When we tested the order-within-color formulations on our small test problem, the solutions to the linear programming relaxations had a savings of 86, and contained a relatively

Formulation	*LP Solution Value*
SVF	199
SIF	199
$DSIF$	152.6
$SP3$	119.2
$SP3$ + 2-subtour	104.8
$SP3$ + 2,3-subtour	101.93
$SP3$ + 2,3,4-subtour	101.87
$ODSIF$	86*
$ODSP3$	86*

Table 2.1: LP Bounds from Formulations of Small Test Problem (*Optimal Value = 86)

small number of fractional variables. CPLEX branch-and-bound (using the best-estimate method for node selection and strong branching for variable selection) was able to find an integer optimal solution, also of savings 86, within 7 minutes. When we added the subtour breaking constraints, the solution to the linear programming relaxation remained at 86, but CPLEX branch-and-bound needed 15 minutes to find the integer optimal solution.

Table 2.1 summarizes our computational experience with various formulations of the small test problem.

2.3.2 Valid Inequalities for the $OSP3$ Formulation

In testing the $OSP3$ formulation, we observed that for many problems, the optimal value of the linear programming relaxation is also the optimal objective value of the integer program, although $OSP3$ does not return an integral solution. However, this equality of objective values is not guaranteed. Consider the example displayed in Figure 2-6.

In the optimal solution of the linear programming relaxation, seven variables (x_{121}, x_{898}, x_{373}, x_{376}, x_{124}, x_{285}, and x_{896}) have value 1/2, for a total of 3.5. Obviously, no integer pro-

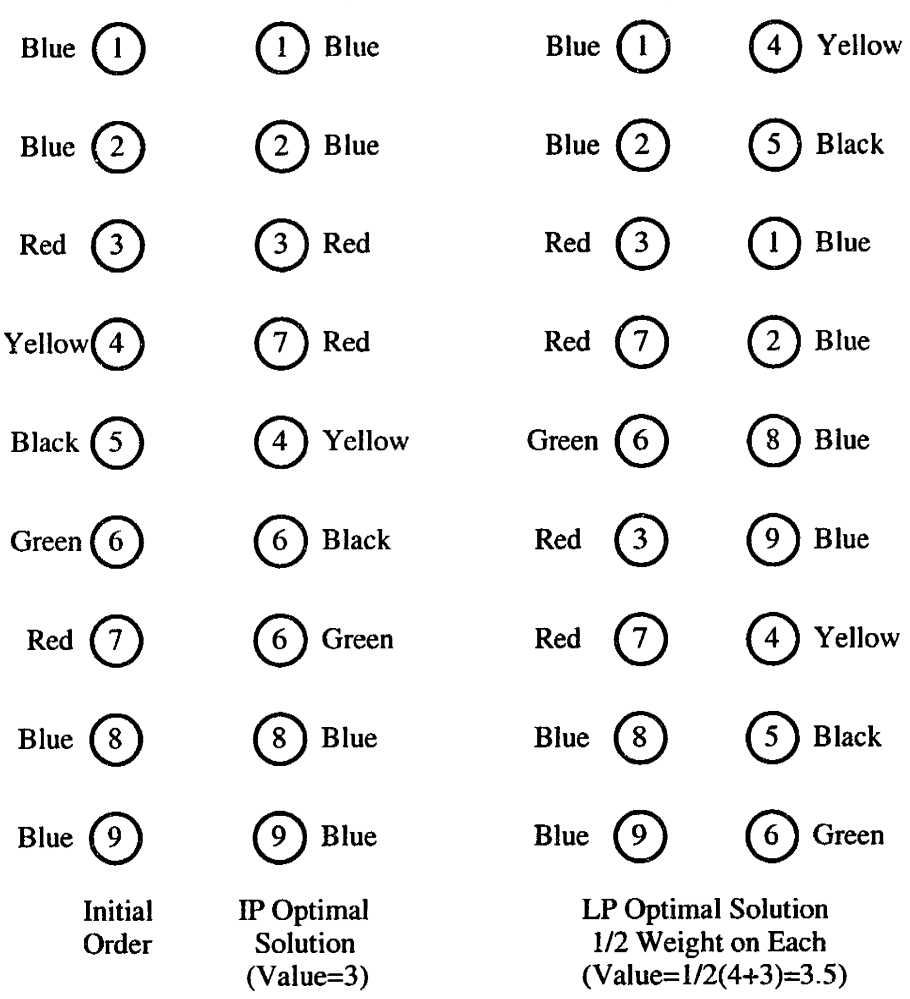

Figure 2-6: Example With Fractional Objective Value (Window Size = 3)

gramming solution can have a fractional objective value, so the optimal integer programming objective value must be at most 3.

The existence of a fractional optimal objective value is due to paint blocks "competing" for space in slots 3 through 7. Due to the time window constraints, the block consisting of consecutive red cars 3 and 7 can appear only in slots 3 and 4, 4 and 5, 5 and 6, or 6 and 7. The block consisting of consecutive blue cars 1, 2, 8, and 9 can appear only in slots 3 through 6 or slots 4 through 7.

Between slots 3 and 7, any sequence has a total of 4 slot transitions (3 to 4, 4 to 5, 5 to 6, and 6 to 7), so the natural bound on transitions between cars of the same color in that range is 4. The blue paint block, because it contains 4 cars, can contribute 3 such transitions in the range $[3, 7]$, and the red block, which contains 2 cars, can contribute 1 transition.

However, if we were to fill slots 3 through 7 entirely with red and blue cars, we would need to make at least one transition between cars of a different color (either red to blue or blue to red) in the slot interval $[3, 7]$, leaving only 3 possible transitions between cars of the same color. Obviously, since there are 5 slots in the interval and no color has more than 4 cars, the interval cannot be filled entirely by cars of one color.

Therefore, we can define the following valid inequality:

$$x_{123} + x_{124} + x_{284} + x_{285} + x_{895} + x_{896} + x_{373} + x_{374} + x_{375} + x_{376} \leq 3.$$

This inequality restricts the number of transitions in the slot interval $[3, 7]$ between two cars of the same color to at most 3. In the fractional solution to the example of Figure 2-6, five of these variables (x_{373}, x_{376}, x_{124}, x_{285}, and x_{896}) have value 1/2, for a total of 2.5, so

the cut is ineffective.

However, we can strengthen the cut in the following manner. We know that the red paint block (cars 3 and 7) and the blue paint block (cars 1, 2, 8, and 9) are restricted to slots 3 through 7 because of the time window constraints. Therefore, we can add all transitions between these cars that occur outside of the slot interval $[3, 7]$ to the left-hand side of the inequality, because no matter where the transitions occur, it is impossible to get all 3 blue-blue transitions and the one red-red transition in the same integer solution.

The new, stronger valid inequality becomes:

$$x_{121} + x_{122} + x_{123} + x_{124} + x_{284} + x_{285} + x_{895} + x_{896} + x_{897} + x_{898} + x_{373} + x_{374} + x_{375} + x_{376} \leq 3.$$

In the fractional solution, seven variables (x_{121}, x_{898}, x_{373}, x_{376}, x_{124}, x_{285}, and x_{896}) have value $1/2$, for a total of 3.5, which violates the cut. In fact, adding this cut to the formulation reduces the optimal objective value to 3, which is also the optimal integer objective.

Using the same ideas, we can define a general class of inequalities that will eliminate fractional linear programming solutions of the type shown in Figure 2-6.

Consider a sequence P of $|P| = l \geq 2$ cars $C_1, ..., C_l$ of the same color. By the order-within-color property, we know that these l cars must appear in order in any solution. Given a window size W, we know that if the l cars appear in consecutive slots, the sequence can start no earlier than slot $a_P = \max\{C_l - W - l + 1, 1\}$. The first term represents the earliest car C_1 can appear given that car C_l can appear no earlier than slot $C_l - W$ and that $l - 2$ cars must appear between C_1 and C_l in the consecutive sequence. The second term bounds the sequence against the first slot. Similarly, the sequence can end no later than slot

$b_P = \min\{C_1 + W + l - 1, N\}$.

Because of the order-within-color property, we know that $C_l \geq C_1 + (l-1)$. So if $a_P + (l-1) \leq b_P$, then we can assign l consecutive slots to the cars of P so that each car is within its allowable window. If this is the case, we call P a "potential block" with block window $W_P = [a_P, b_P]$. If it is impossible for the last car of a potential block P to be the first car of another potential block of the same color (thus it is "out of range" of the next car of the same color) and it is impossible for the first car of a potential block to be the last car of another potential block of the same color (thus it is out of range of the previous car of the same color), then we call P a "separated" potential block.

Suppose we are given two slots A and B with $A < B$, and 2 colors, each with a separated potential block P_k whose block window falls within the interval $[A, B]$. Thus $A \leq a_{P_k} < b_{P_k} \leq B$ for each color k. We assume that $|P_1| + |P_2| \geq B - A + 1$; otherwise, our cut will have no effect.

Define a "free transition" as two consecutive slots containing cars of the same color. How many free transitions could the interval $[A, B]$ contain? Suppose a sequence has m free transitions of color 1. Then it can have at most $B - A - m - 1$ free transitions of color 2, because the interval $[A, B]$ contains a maximum of $B - A$ free transitions, minus the m free transitions of color 1, minus 1 transition (which will not be free) representing the switch from color 1 to color 2 (or vice versa).

Thus in our formulation, we can bound the sum of the variables corresponding to these free transitions by $B - A - 1$. Moreover, because we must split up at least one of these complete potential blocks in order to do any better with the other complete potential block,

we can bound the sum of *all* transitions between these cars by the same value, even if they occur outside of the range $[A, B]$.

We can generalize these inequalities to apply to K colors, with a bound of $B - A - K + 1$ on the sum of the transitions.

$$\sum_{k=1}^{K} \sum_{l=1}^{|P_k|-1} \sum_{t=1}^{N-1} x_{C_{l-1}^k C_l^k t} \leq B - A - K + 1.$$

These inequalities are explicit forms of the *composed path inequalities* described by Balas and Schulz [5]. In general, there are many of these inequalities, they take a long time to generate, and their contribution to the linear programming bound is small; thus, we normally omit them from our formulation.

In general, calculating the dimension of a TSPTW is NP-hard; therefore, deciding whether an inequality is a facet of the integer polyhedron is a difficult problem. For the example of Figure 2-6, we used the PORTA software package [8] to determine that the inequality is indeed a facet of the $SP3$ polyhedron. Without the inequality, the dimension of the polyhedron is 178; with the inequality, the dimension is 177.

2.3.3 Solving the Linear Programming Relaxation of $OSP3$ using Lagrangean Relaxation

Although the formulation $OSP3$ is usable for larger problems than the polynomial-time DP, it still is too large for the original instance. For the 750-car, $w = 75$, 15-color case, the formulation contains over $14,000,000$ variables. However, using Lagrangean Relaxation

and exploiting the structure of the objective function, we can solve the linear programming relaxation of the 14,000,000-variable problem quickly.

Our time-indexed formulation can be thought of as a shortest-path network with N side constraints, one for each car, specifying that each car must be used exactly once. Without those side constraints, the formulation becomes a shortest path problem on an acyclic graph, which can be solved in $O(E)$ time as a function of the number E of edges of the graph [2].

Thus, we can define a Lagrangean relaxation by associating a vector of Lagrange multipliers μ with the constraints stating each car must be preceded by exactly one other car.

For convenience, we define sets S_{OWC} and $\hat{S}_{OWC}$ to be all the elements of sets S and $\hat{S}$ (as defined for the $SP3$ formulation) except those prohibited by the order-within-color property. We can then solve the Lagrangean subproblem

$$\text{maximize} \quad \sum_i x_{i,\text{cnext}(i),t} - \sum_j \mu_j (1 - \sum_{(i,j)\in \hat{S}_{OWC}(t)} x_{ijt})$$

$$\begin{aligned}
\text{subject to} \quad \sum_{j\in S_{OWC}(t+1)} x_{ijt} - \sum_{k\in S_{OWC}(t-1)} x_{kit} &= 0, \\
&\forall (i,t) : i \in S_{OWC}(t) \\
\sum_{j\in S_{OWC}(1)} x_{s^* j1} &= 1 \\
\sum_{i\in S_{OWC}(N)} x_{it^* N} &= 1 \\
x_{ijt} &\in \{0,1\}, \\
&\forall (i,j,t) : (i,j) \in \hat{S}_{OWC}(t)
\end{aligned}$$

as a shortest path problem. Moreover, because the associated network is acyclic, we can solve the problem by examining each edge once. Using this shortest path subproblem in a Lagrangean relaxation procedure, we can find a relaxation of the optimal solution to the integer program; further, because the solution to the Lagrangean subproblem is integral (because it is an integer network optimization problem), the value of the Lagrangean relaxation will be exactly equal to the value of the linear programming relaxation.

For a problem with $14,000,000$ variables, even examining each edge can take a prohibitively long amount of time. Each iteration takes approximately 5 seconds, meaning that 5000 iterations of the Lagrangean procedure would take almost 7 hours to complete.

However, by taking advantage of the structure of the problem, we can reduce the running time by a factor of 50. If we solved the shortest path problem by a standard algorithm, we could, for each node (i, t) of the network (i is the car and t is the time slot), find the minimum distance from any of the eligible nodes $(j, t-1)$ corresponding to assigning car j to slot $t-1$ and car i to slot t. The underlying network can contain up to $2W$ such nodes, requiring up to $2W - 1$ comparisons to determine the minimum.

Observe that, since the Lagrangean term in the objective for any edge depends only on the car associated with the tail of the edge, for any time slot t, the minimum-cost incoming edge will be the same for every car i unless the bonus for the edge from its within-color predecessor makes that edge more desirable. Therefore, for every time slot t we need only compare all $2W + 1$ possible incoming edges once (without any color bonuses), and then for each car we make only one comparison, between the generic minimum edge and the previous car of the same color (if feasible).

Also notice that in some cases, we can cut down the number of comparisons from one time slot to the next. For every node (i, t), the incoming edges are all from the same set of cars (and use the same set of multipliers) and for node $(i, t+1)$

Using these facts, we were able to reduce the solution time of the Lagrangean subproblem so that we could solve each $14,000,000$-variable problem in 0.1 seconds; thus, 5000 iterations of the Lagrangean procedure takes about $8\frac{1}{3}$ minutes instead of 7 hours.

2.4 Block-Enumerative Formulations

In a general TSP or TSPTW, it is difficult to enumerate all potential sequences or subsequences of cities. The smallest case, enumerating all potential subsequences of size 1, is the standard formulation (n such subsequences in an n-city problem); however, enumerating all potential subsequences of size m for an n-city problem involves $\frac{n!}{(n-m)!}$ possibilities.

However, in our problem, two factors make enumeration a viable alternative formulation. First, the cities (cars) are partitioned by color, so we can consider only subsequences of identically-colored cars. Second, because of the order-within-color property, we know the relative sequence of all cars within each color; furthermore, the cost structure ensures that the order-within-color solution corresponding to any feasible solution will have the same objective value as that feasible solution. Therefore, we can consider the sequence of cars within each color, and enumerate all feasible (based on the time window constraints) ordered subsequences (paint blocks).

We define B to be the set of all paint blocks that obey the time window constraints. For every paint block $b \in B$, we define the size s_b of the block to be the number of cars it

contains.

Because of the order-within-color property, at most $N - i + 1$ blocks can begin with the ith car; therefore, there are at most $\frac{N^2+N}{2}$ possible blocks. For the real-world problems we address, we have a maximum of of $281,625$ blocks, and because of the time window restrictions and the multiple colors, the true number $|B|$ of possible blocks is between ten and eleven thousand. By comparison, the potential number of blocks without the order-within-color property could be as high as $O(10^{150})$ for the real-world problems we are considering.

2.4.1 Complete Enumerative Formulation

For every block $b \in B$, the block size s_b and the time window constraints define the earliest slot l_b and the latest slot u_b in which block b can begin. For example, suppose block b contains cars 3, 5, and 8. If the window size w is 2, then block b can begin in slot 4 (car 3 in slot 4, car 5 in slot 5, and car 8 in slot 6) or slot 5 (car 3 in slot 5, car 5 in slot 6, and car 8 in slot 7). Block b cannot begin earlier than slot 4, because then car 8 would be forced to move outside of its allowable range $[6, 10]$, and similarly block b cannot begin later than slot 5 without forcing car 3 outside of its allowable range $[1, 5]$. Therefore, $l_b = 4$ and $u_b = 5$.

We define a zero/one variable x_{bj} for every possible starting slot j of every block $b \in B$,

and create a complete enumerative formulation (CEF).

$$
\begin{aligned}
[CEF:] \text{ maximize} \quad & \sum_{b \in B} \sum_{j: l_b \le j \le u_b} (s_b - 1) x_{bj} & & & \\
\text{subject to} \quad & \sum_{b \in B: i \in b} \sum_{j: l_b \le j \le u_b} x_{bj} & = & 1 & \forall \text{ cars } i \\
& \sum_{b \in B} \sum_{j: l_b \le j \le u_b, j \le k \le (j + s_b - 1)} x_{bj} & = & 1 & \forall \text{ slots } k \\
& x_{bj} & \in & \{0,1\} & \forall \text{ blocks } b, \forall \text{ slots } j.
\end{aligned}
$$

The first set of constraints ensures that every car i is assigned a slot in the paint-blocking scheme, by summing all variables corresponding to blocks that contain car i. The second set of constraints ensures that every slot k is assigned a car, by summing all variables corresponding to an assignment of a block to a range of slots that includes k.

Since every block b contains s_b cars, $s_b - 1$ of the cars in block b will follow a car of the same color. Therefore, the objective function minimizes the number of paint blocks by counting the number of cars whose predecessor is the same color.

This formulation will have $2N$ constraints, one for each car and one for each slot. It can contain more than $WN(N+1)$ variables, since each of the up to $\frac{N^2+N}{2}$ blocks can start in up to $2W+1$ slots. Moreover, the constraint matrix will likely be dense, because each block b can begin in $u_b - l_b + 1$ different slots. In our real-world instance, the formulation contains nearly $900,000$ variables and the constraint matrix contains over $11,000,000$ nonzeros.

2.4.2 Time-Relaxed Formulation

The complete enumerative formulation is an exact formulation for the problem. We can obtain an upper bound on the optimal objective value by relaxing (or in this case, removing) some of the constraints.

If we eliminate the timing constraints (the second set of constraints), the columns of the new constraint matrix will be highly dependent. Specifically, for any block b, the columns corresponding to x_{bj} are identical for any feasible j. Thus, for every block b, we can replace all of the variables x_{bj} with a single zero/one variable y_b to obtain a time-relaxed formulation $TREL$.

$$
\begin{aligned}
[TREL:] \text{ maximize } & \sum_{b \in B} (s_b - 1) y_b \\
\text{subject to } & \sum_{b \in B : i \in b} y_b = 1 \qquad \forall \text{ cars } i \\
& y_b \in \{0,1\} \quad \forall \text{ blocks } b \in B.
\end{aligned}
$$

Moreover, because each constraint of $TREL$ contains variables for blocks of only a single color, $TREL$ fully decomposes by color. If we define B_c as the set of blocks of color c, we can decompose the relaxed formulation into C subproblems of the following form:

$$
\begin{aligned}
[TREL_c:] \text{ maximize } & \sum_{b \in B_c} (s_b - 1) y_b \\
\text{subject to } & \sum_{b \in B_c : i \in b} y_b = 1 \qquad \forall \text{ cars } i \text{ of color } c \\
& y_b \in \{0,1\} \quad \forall \text{ blocks } b \in B_c.
\end{aligned}
$$

We can think of removing the timing constraints as a type of Lagrangean relaxation

procedure in which we set all of the Lagrangean multipliers to zero.

For our real-world instances, the undecomposed problem has over 10,000 variables and 750 constraints, and decomposes into 15 problems of various size.

An Optimal Greedy Algorithm for the Decomposed Problem

We can quickly solve each subproblem $TREL_c$ to optimality using a greedy algorithm.

Optimal Greedy Algorithm for $TREL_c$

While some car i of color c is not assigned to any block

    Assign the smallest-indexed unassigned car of color c to a new block

    While the next unassigned car of color c can feasibly be in the new block

        Assign that car to the new block

    End while

End while

We can prove that this greedy algorithm is optimal as follows. We first note that the order within color property applies to the time-relaxed problem. That is, the argument we have used before shows that the time-relaxed problem has a solution satisfying the order-within-color property.

To show that the greedy algorithm solves the time-relaxed problem, let S be any feasible solution to the time-relaxed problem satisfying the order-within-color property. Suppose that the greedy solution G and the solution S agree for the first i cars of color c in the sense that the first i cars in S and G belong to the same blocks. Suppose that the greedy solution

G and the feasible solution S do not agree for the first $i+1$ cars of color c. Then the $(i+1)$st car of color c must belong to block b in the greedy solution, and to some other block b' in the solution S. The provisions of the greedy algorithm show that we can feasibly modify the solution S by moving the $(i+1)$st car from block b' to block b, creating a new solution S'. S' has no more blocks than S. If we repeat the procedure, we eventually transform the solution S to the greedy solution and show that the greedy solution contains no more blocks than the solution S. Since we chose the solution S as an arbitrary feasible solution to the time-relaxed problem satisfying the order-within-color property, the greedy solution solves the time-relaxed problem.

2.5 Comparison of Strong Formulations

2.5.1 Upper Bound Generation

As a measure of the quality of any feasible integer solution to the problem, we would like to compare the solution value with a provable upper bound on the objective value. We have seen four formulations whose linear programming relaxations tend to give strong upper bounds: the assignment formulation with indicator variables ($ODSIF$), the time-indexed formulation with order-within-color restrictions ($OSP3$), the complete block-enumerative formulation (CEF), and the time-relaxed block-enumerative formulation ($TREF$).

For the real-world test case, we can not solve either the block-enumerative formulation or the $OSP3$ formulation relaxations directly using a standard linear programming solver; in both cases, the computer memory requirements just to store the problems exceed the

Formulation	*Constraints*	*Variables*	*Nonzeros*	*Method*	*Upper Bound*	*Time*
CEF	1500	900,000	11,000,000	None	—	—
ODSIF	200,000	200,000	700,000	Simplex	681	48 hours
ODSIF	200,000	200,000	700,000	Barrier	681	20 minutes
*OSP*3	200,000	14,000,000	43,000,000	Lagrangean	681	8.3 minutes
TREL	750	10500	100,000	Greedy	681	< 0.5 seconds

Table 2.2: Upper Bounds from Formulations of Real-World Test Case

available 384 megabytes of RAM (an extremely large amount). However, the specialized Lagrangean relaxation method can generate an upper bound from the $OSP3$ formulation in less than 10 minutes.

We could solve the linear programming relaxation of the $ODSIF$ formulation directly using the simplex algorithm; however, the running time is measured in days. In contrast, the barrier method is able to generate the upper bound being in approximately 20 minutes. We suspect that the time difference is due to the sparsity of the matrix; because of the many indicator constraints, the density of the matrix is less than 0.00002.

The greedy algorithm solves the time-relaxed block-enumerative formulation in less than one half of one second.

Table 2.2 summarizes our computational experience with the strong formulations (formulations that give a strong upper bound) for the real-world test case.

2.5.2 General-Case Bound and Feasibility Comparisons

As shown in Table 2.2, each relaxation we were able to solve has an optimal objective value of 681. In this section, we show that although the relaxations of the three complete formulations

CEF, $ODSIF$, and $OSP3$ have different feasible sets, they will always generate the same upper bound.

Comparison of $OSP3$ and $ODSIF$

Any feasible solution $\hat{x}$ to the relaxation of $OSP3$ has a corresponding feasible solution $(\bar{x}, \bar{I})$ to the relaxation of $ODSIF$.

For any $\hat{x}$ that is feasible for the relaxation of $OSP3$, we can define the following feasible solution to the relaxation of $ODSIF$:

$$\bar{x}_{it} = \sum_j \hat{x}_{ijt},$$

$$\bar{I}_{it} = \hat{x}_{i(cnext(i))t}$$

For $(\bar{x}, \bar{I})$ to be feasible, it must satisfy the constraints of the relaxation of $ODSIF$.

Since $\hat{x}$ is a feasible solution to the relaxation of $OSP3$, $\sum_j \sum_t \hat{x}_{ijt} = 1$ for all j and t. Therefore,

$$\sum_t \bar{x}_{it} = \sum_t (\sum_j \hat{x}_{ijt}) = 1 \ \forall i.$$

We will prove that $\sum_i \bar{x}_{it} = 1$ for all t by induction.

$$\sum_i \bar{x}_{i0} = \sum_i (\sum_j \hat{x}_{ij0}) = \sum_i (\hat{x}_{s^*i}) = 1.$$

Assuming now that $\sum_i \bar{x}_{it} = 1$ is true for $t = \tau - 1$,

$$\sum_i \bar{x}_{i\tau} = \sum_i (\sum_j \hat{x}_{ij\tau}) = \sum_i (\sum_j \hat{x}_{ji(\tau-1)}) = \sum_j (\sum_i \hat{x}_{ji(\tau-1)}) = \sum_j \bar{x}_{j(\tau-1)} = 1.$$

Finally,

$$\bar{I}_{it} = \hat{x}_{i(cnext(i))t} \leq \sum_j \hat{x}_{ijt} = \bar{x}_{it}, \text{ and}$$

$$I_{it} = \hat{x}_{i(cnext(i))t} \leq \sum_j \hat{x}_{j(cnext(i))t} = \sum_j \hat{x}_{(cnext(i))j(t+1)} = \bar{x}_{(cnext(i))(t+1)}.$$

Therefore, the feasible solution $\hat{x}$ to the relaxation of $OSP3$ has a corresponding feasible solution $(\bar{x}, \bar{I})$ to the relaxation of $ODSIF$. Because there is a one-to-one correspondence between the values of $\hat{x}_{i(cnext(i))t}$ and $\bar{I}_{it}$, the objective values of these two solutions will be the same.

On the other hand, a feasible solution $(\bar{x}, \bar{I})$ to the relaxation of $ODSIF$ does not determine a unique feasible solution $\hat{x}$ to the relaxation of $OSP3$; instead, each $(\bar{x}, \bar{I})$ might define an infinite number of $\hat{x}$s. Furthermore, the objective value of $(\bar{x}, \bar{I})$ might be worse than the objective value of any corresponding $\hat{x}$. However, the objective value of an optimal $(\bar{x}, \bar{I})$ will be the same as in its corresponding $\hat{x}$s.

Consider some feasible $(\bar{x}, \bar{I})$ for the relaxation of $ODSIF$. The value of $\bar{x}_{it}$ defines how much flow enters and leaves node (i, t) in the network of the relaxation of $OSP3$, but does not specify specific values of $\hat{x}_{ijt}$. For each time t, we can solve a bipartite flow matching problem on the nodes $(i, t-1)$ and (i, t) for all i to define feasible values of $\hat{x}_{ijt}$. Because many feasible bipartite flow matchings might be possible, there can be many corresponding feasible solutions $\hat{x}$, each conceivably with a different objective value. However, each feasible bipartite flow matching defines a feasible solution to the relaxation of $OSP3$; the flow balance constraints are automatically satisfied by the flow matching, and $\sum_{j,t} \hat{x}_{ijt} = \sum_t (\sum_j \hat{x}_{ijt}) = \sum_t \bar{x}_{it} = 1$.

To attempt to obtain a bipartite flow matching with the same objective value as $(\bar{x}, \bar{I})$, we can specify that $\hat{x}_{i(cnext(i))t} = \bar{I}_{it}$ for each $\bar{I}_{it}$, and solve a bipartite flow matching problem on the remaining flows.

It is possible, for a general feasible solution $(\bar{x}, \bar{I})$, that there is no feasible matching once the value of $\hat{x}_{i(cnext(i))t}$ is set. For example, suppose that $\bar{x}_{it} = \bar{x}_{(cnext(i))(t+1)} = 1$, and $\bar{I}_{it} < 1$. After setting $\hat{x}_{i(cnext(i))t} = \bar{I}_{it}$, there are no feasible arcs on which to flow the remaining $1 - \bar{I}_{it}$ units from node (i, t). Therefore, a feasible solution to the relaxation of $ODSIF$ might not have a corresponding feasible solution to the relaxation of $OSP3$ with the same objective value.

However, if the solution to the relaxation of $ODSIF$ is "I-maximal", then a corresponding feasible solution to the relaxation of $OSP3$ with equal objective value will exist. A I-maximal solution is a feasible solution $(\bar{x}, \bar{I})$ to the relaxation of $ODSIF$ with $\bar{I}_{it} = \min(\bar{x}_{it}, \bar{x}_{(cnext(i))(t+1)})$. The corresponding solution $\hat{x}$ to a I-maximal solution $(\bar{x}, \bar{I})$ will set $\hat{x}_{i(cnext(i))t} = \bar{I}_{it}$, so either node (i, t) or node $(cnext(i), t+1)$ or both will have zero remaining flow in the bipartite flow matching problem for time t. Thus, the flow problem will remain feasible, and a full solution $\hat{x}$ with the same objective value as $(\bar{x}, \bar{I})$ can be found.

It is easy to show that every optimal solution to the relaxation of $ODSIF$ must be I-maximal. Suppose that $\bar{I}^*_{it} < \min(\bar{x}^*_{it}, \bar{x}^*_{(cnext(i))(t+1)})$ in some optimal solution $(\bar{x}^*, \bar{I}^*)$ for some (i, t). Then, without changing any other variables, we could increase $\bar{I}^*_{it}$ by some small value ϵ ($\epsilon <= \min(\bar{x}^*_{it}, \bar{x}^*_{(cnext(i))(t+1)}) - \bar{I}^*_{it}$), thereby increasing the objective value by ϵ and contradicting the optimality of $(\bar{x}^*, \bar{I}^*)$.

Therefore, every optimal solution to the relaxation of $ODSIF$ has a corresponding feasible solution to the relaxation of $OSP3$ with the same objective value. Since we previously showed that any feasible solution for the relaxation of $OSP3$ has a corresponding solution to the relaxation of $ODSIF$ with the same objective value, we have shown that the relaxation of $OSP3$ is a stronger formulation in the sense of feasibility, but that both formulations will provide the same upper bound on the integer programming optimal solution value.

Comparison of $OSP3$ and CEF

Any feasible solution $\hat{x}$ to the relaxation of $OSP3$ has at least one corresponding feasible solution $\tilde{x}$ to the relaxation of CEF with the same objective value.

Suppose we are given any feasible $\hat{x}$. We can decompose $\hat{x}$ into a set P of paths in the the relaxation of the $OSP3$ network from the source node s^* to the sink node t^*. Let f_p be the amount of flow on some path $p \in P$. Each path p decomposes into maximal sets of consecutive same-colored cars along the path, corresponding to blocks b beginning at times t, where t is the slot in the path in which the first car of the block is visited. Therefore, for each path p, we can add f_p to $\tilde{x}_{bt}$. Performing this operation on all paths of the decomposition gives a final solution $\tilde{x}$ to the relaxation of CEF.

By definition, $\tilde{x}_{bt} = \sum_{p:b,t\in p} f_p$, so

$$\sum_{b,t:i\in b} \tilde{x}_{bt} = \sum_{b,t:i\in b} \sum_{p:b,t\in p} f_p = \sum_{p:i\in p} f_p = 1$$

since $\sum_{p:i\in p} f_p$ counts the total flow through car i, which is equal to $\sum_{j,t} \hat{x}_{ijt} = 1$.

Also,

$$\sum_{b,t:\tau\in[t,t+s_b-1]} \tilde{x}_{bt} = \sum_{b,t:\tau\in[t,t+s_b-1]} \sum_{p:b,t\in p} f_p = \sum_{p\in P} f_p = 1$$

since every path p must pass through slot τ exactly once.

Therefore, $\tilde{x}$ is feasible for the relaxation of CEF.

The objective value of $\hat{x}$ equals $\sum_{i,t} \hat{x}_{i(cnext(i))t}$. For each path p, $\sum_{b,t\in p}(s_b - 1)\tilde{x}^p_{bt} = \sum_{i,t\in p} \hat{x}^p_{i(cnext(i))t}$. Therefore, the objective value of the relaxation of CEF is

$$\sum_p \sum_{b,t\in p} (s_b - 1)\tilde{x}^p_{bt} = \sum_p \sum_{i,t\in p} \hat{x}^p_{i(cnext(i))t} = \sum_{i,t} \hat{x}_{i(cnext(i))t}$$

and so the two objectives are the same.

Any feasible solution $\tilde{x}$ to the relaxation of CEF has at least one corresponding feasible solution to the relaxation of $OSP3$, although the objective functions might be different. Every $\tilde{x}_{bt} > 0$ contributes to $\tilde{x}_{bt}$ to the values of $\hat{x}_{i(cnext(i))\tau}$ for every $(i, cnext(i), \tau) \in (b, t)$. We can create the remaining elements of $\hat{x}$ by solving one bipartite flow matching problem for each time t, as in the case of creating a solution to the relaxation of $OSP3$ from a solution to the relaxation of $ODSIF$. As before, the resulting solution $\hat{x}$ will be feasible for the relaxation of $OSP3$.

However, the objective value of $\hat{x}$ might be greater than the objective value of $\tilde{x}$. Suppose $\tilde{x}$ has two consecutive blocks of the same color. Those two blocks b_1 and b_2 will contribute $s_{b_1} - 1 + s_{b_2} - 1$ to the objective function of the relaxation of CEF; however, because the relaxation of $OSP3$ will just see $s_{b_1} + s_{b_2} - 1$ consecutive edges of the form $\hat{x}_{i(cnext(i))t}$, it will have an objective value that is one greater.

Define an $\tilde{x}$-maximal solution to be a solution to the relaxation of CEF in which every block is maximal. Therefore, in the example above, blocks b_1 and b_2 could not exist consecutively; they would be combined into one block.

In the case of an $\tilde{x}$-maximal solution, every within-block transition will still correspond to one edge of the form $\hat{x}_{i(cnext(i))t}$. No other transitions in $\tilde{x}$ will be between cars of the same color, so the values of every $\hat{x}_{i(cnext(i))t}$ will be set only by within-block transitions. Thus, an $\tilde{x}$-maximal solution to the relaxation of CEF has a corresponding solution $\hat{x}$ to the relaxation of $OSP3$ with the same objective value.

We have shown that any optimal solution $\hat{x}$ to the relaxation of $OSP3$ has a corresponding optimal solution $\tilde{x}$ to the relaxation of CEF with the same objective value, and vice versa. Previously, we have shown the same relationship between optimal solutions of the relaxations of $OSP3$ and $ODSIF$. Therefore, the relaxations of all three formulations $ODSIF$, CEF, and $OSP3$ will provide the same upper bound on the value of the optimal integer solution.

2.5.3 Dynamic Programming Formulation

Because the order-within-color property ensures that we can always create a position-preserving reordering for each color, we can use the following dynamic programming formulation. Let $J_n(c, n_1, n_2, \ldots, n_{C-1})$ denote the cost-to-go when (i) the first n slots have been filled by n_1 cars of color 1, n_2 cars of color 2, ..., n_{C-1} cars of color $C-1$, and $n - \sum_{i=1}^{C-1} n_i$ cars of color C, and (ii) the last slot filled contains a car of color c. $J_n(c, n_1, n_2, \ldots, n_{C-1}) = \min\{J_{n-1}(1, n_1-1, n_2, \ldots, n_{C-1}), J_{n-1}(2, n_1, n_2-1, \ldots, n_{C-1}), \ldots, J_{n-1}(C-1, n_1, n_2, \ldots, n_{C-1} - 1), J_{n-1}(C, n_1, n_2, \ldots, n_{C-1})\}$.

Feasibility depends only on whether the n_ith car of color i is eligible (by its time window constraints) to appear in slot $n+1$.

At each stage we are essentially dividing n slots among a maximum of C colors (it might not be feasible for some colors to appear in the first n slots), so we can have no more than $\binom{n+C-1}{n} C$ dynamic programming nodes at stage n. Since $C > 1$, $\binom{(n+1)+C-1}{n+1} > \binom{n+C-1}{n}$. Also, for $C > 1$ and $n > 2$, $\binom{n+C-1}{n} < n^{C-1}$. Thus the total number of DP nodes will be less than $N(N^{C-1}C) = CN^C$. So the size of the DP tree will be $O(N^C)$, which is polynomial given the assumption of the fixed, small size C of our color set. Our real-world data specifies a set of 15 possible colors; thus, the DP algorithm in that case will run in $O(N^{15})$ time.

The preceding algorithm is essentially the same as the dynamic programming recursion developed by Tsitsiklis [29] for instances of the TSPTW with a small number of cities to be visited, but where each city might need to be visited more than once. Because of the order-within-color property, we can consider each color to be a city. We need to visit a city once for each car of its color, with an appropriate time window. We define the travel time between any two cities to be one (even between a city and itself) for time-window purposes, and assign a cost of 1 for traversing an intercity link and a cost of 0 for traversing an intracity link.

The fact that we can define a polynomial-time dynamic programming algorithm for fixed C does not ensure that we can use it to solve practically-sized problems. For our problem size with $C = 15$ colors, the polynomial DP algorithm will require $O(N^{15})$ time, with $N = 750$ cars. While the phrase "polynomial time" might seem to imply a fast solution, because

of the large polynomial exponent, the number of potential DP nodes is very large — more than 2^{147} DP nodes for our size of problem. Obviously, this instance is not realistically solvable using the polynomial-time dynamic programming algorithm, and thus we will need to solve the problem in another way. Specifically, we will approach the problem as if it were NP-hard; we will attempt to use our strong formulations to provide good bounds and heuristic approximation methods for our problem.

Chapter 3

Solution Algorithms

As we noted previously, our various exact formulations are too large to solve directly; in fact, even the linear programming relaxation of the CEF formulation is still too large to solve, and solving the other formulations' linear programming relaxations can take hours.

On the surface, this problem seems to be one that is solved daily, and thus a solution that requires an hour or two might be acceptable. However, in a real assembly line, cars do not always proceed straight through the assembly process. Occasionally (several times per day), the plant will not correctly perform some operation and must remove a car from the assembly line, fix it, and then reinsert it later [13] [4] [22]. The result is a disrupted sequence with a potential increase in the number of paint switchovers. To retain large paint blocks, the plant will once again need to reorder the cars. Thus, a heuristic solution process that requires an hour of computation is unacceptable. Ideally, we would like to develop a good heuristic that will run in a few minutes.

3.1 General Improvement Algorithms

Given any feasible solution, we can attempt to improve it by locally optimizing over a small part of the solution. For the general TSPTW, even finding a feasible solution is an NP-hard problem, but for our special case we can construct an obvious starting feasible solution by initially assigning every car to its pre-ordered slot.

Savelsbergh [27] and Kindervater, Lenstra, and Savelsbergh [16] suggest improving a current feasible solution by using k-interchange heuristics similar to those used for the standard traveling salesman problem. We tested interchange heuristics on our problem, but found that they did not provide good solutions to our special case of the TSPTW. Once paint blocks begin to form using interchange heuristics, different blocks will be different sizes. Thus, unless two entire blocks are the same size, interchanging will be difficult because we will need to break one or more of the interchanged blocks. Thus, although interchange heuristics are useful for the general TSP and TSPTW, they tended to provide solutions that were 15-20% from optimal for our problems.

3.2 Within-Window Preprocessing

For convenience, we define a "window" as being a set of consecutive cars or slots numbered $kW + 1$ through $kW + W$, for some nonnegative integer k. We denote this window as the $(k + 1)$st window, so that windows are numbered $1, 2, ..., \frac{N}{W}$.

Notice that within each set of W consecutive cars, we can arbitrarily reorder the cars while retaining feasibility. Therefore, we can split the assembly line into buckets of length W

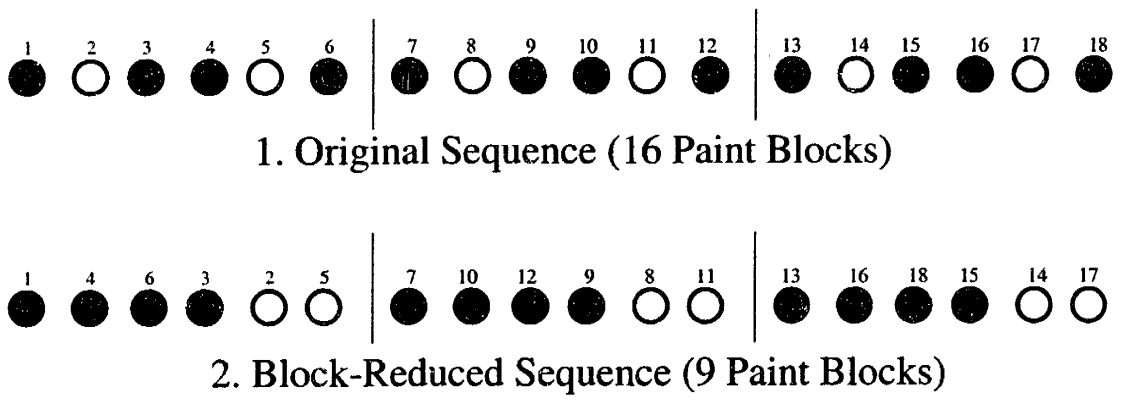

Figure 3-1: Within-Window Preprocessing Algorithm

and locally optimize each one by simply grouping cars of the same color within each bucket (see Figure 3-1). Assuming each bucket contains each color, this gives a total of $\frac{NC}{W}$ blocks, or a total score of $\frac{N(W-C)}{W}$ cars that follow a car of the same color. We found that this score was about 10% from the optimal solution in most of our full-sized problems.

More importantly, this within-window preprocessing algorithm provides us with an opportunity to define a reduced-size TSPTW, reducing the number of "cities" by a factor of $\frac{W}{C}$. To achieve this reduction, we maintain the blocks created by this preprocessing step. That is, instead of looking at individual cars, we now treat all the same-colored cars of a window as a single block by forcing them to stay together in any reordering. The problem is still difficult to solve, and the integer programming formulations (and their linear programming relaxations) are still too large to be useful for large window sizes, but the reduced problem is small enough that we can use other fast heuristics to tackle it.

Forcing this grouping of cars can reduce the achievable optimal solution; in our tests of real-world problems, the difference in LP bounds was never more than 0.5%.

3.3 Two-By-Two Heuristic

Although the overall integer programming formulations still take far too long to solve even after within-window preprocessing, the block enumerative formulation applied to just a two-window set of cars solves quickly using CPLEX branch-and-bound after the initial preprocessing step. We can exploit this fact to create a heuristic solution method, the two-by-two heuristic.

After the preprocessing stage, the two-by-two heuristic uses branch-and-bound to find the optimal reordering of windows k and $k+1$, for all odd k. Therefore, it solves $\frac{N}{2W}$ block enumerative integer programs of size $2W$ cars each, beginning with windows 1 and 2, then windows 3 and 4, then windows 5 and 6, and so forth.

Two-By-Two Heuristic

Begin with the given pre-ordered solution

Run within-window preprocessing

For $k = 1$ to $\frac{N}{W} - 1$ do

    If k is odd

        Solve the optimization problem on the kth and $(k+1)$st window lengths

End for

When running the two-by-two heuristic, we impose the order-within-color restriction at all times.

When we prove some theoretical performance bounds on the two-by-two heuristic in the next chapter, we will find the following result useful: For any odd k, suppose there are p

colors for which all the cars initially in windows k and $k+1$ are in the same block in the optimal solution. Then, the two-by-two heuristic on windows k and $k+1$ can create a solution which is at least as good as creating a single block of each of those same p colors.

To prove this property, we will show which slots certain color blocks can appear in, and use this to prove our result.

Consider the optimization for windows k and $k+1$. After preprocessing, each window will have at most one block per color. Therefore, the two-by-two heuristic can make improvements only by matching two blocks of the same color, one from window k and one from window $k+1$, so that they appear consecutively in the optimized solution. By examining the possible beginning and ending locations of these consecutive blocks, we can prove that all pairs of consecutive same-colored blocks from windows k and $k+1$ (for odd k) that appear in the global optimal solution to the preprocessed problem can also appear in the two-by-two heuristic solution to the preprocessed problem.

We begin by observing the range of slots in which consecutive same-colored blocks can begin in the global optimal solution.

Lemma 1 *Consider any block consisting of cars from windows k and $k+1$. Suppose the block starts in some slot l in any feasible solution (including the optimal solution). If slot l is in window k or $k-1$, then the block can also feasibly start in any slot between l and the first slot of window $k+1$.*

Proof: Since the first car of the block is from window k (or from window $k+1$), its feasible slots extend up to window $k+1$. Now consider the ith car in the block, for $i > 1$. By the order-within-color property, the ith car in the block must be at least $i-1$ cars higher

in the initial sequence than the first car. Therefore, if the first car can reach some slot m in the $k+1$st window, the ith car can reach at least $i-1$ slots higher, and so the block can begin in the first slot of window $k+1$. □

Lemma 2 *Consider any block consisting of cars from windows k and $k+1$. Suppose the block ends in some slot l in the optimal solution. If slot l is in window $k+1$ or $k+2$, then the block can also feasibly end in any slot between l and the last slot of window k.*

Proof: This proof is similar to the proof of the previous lemma. Since the last car of the block is from window $k+1$ (or from window k), its feasible slots extend down to window k. Now consider the ith car in the block, for $i <$ the block size. By the order-within-color property, the ith car in the block must be at least the block size minus i cars lower in the initial sequence than the last car. Therefore, if the last car can reach some slot m in the kth window, the ith car can reach at least the block size minus i slots lower, and so the block could end in the last slot of window k. □

Using Lemmas 1 and 2, we can now show that all consecutive same-colored blocks in the global optimal solution can also appear in the two-by-two heuristic solution.

In any feasible solution, we refer to a color c as *paired* with respect to windows k and $k+1$ if the blocks of color c from windows k and $k+1$ appear together as a signle block. Note that the paired colors might not appear entirely in windows k and $k+1$; it might be possible for the block to begin in window $k+1$ and end in window $k+2$, or to begin in window $k-1$ and end in window k.

Theorem 1 *For any odd k, suppose that some feasible solution (e.g., the optimal solution), has p paired colors with respect to windows k and $k+1$. Then, the two-by-two heuristic on*

windows k and $k+1$ can create a solution that is at least as good with the same p paired colors now appearing in windows k and $k+1$.

Proof: We can prove this theorem constructively, by induction on k.

Consider the locations in the optimal solution of every car from windows 1 and 2. Due to the window size, only cars from windows 1 and 2 can belong to slots in window 1.

Now consider any cars initially in windows 1 or 2 that are in windows 2 or 3 in the optimal solution. (Because of the window size, they cannot be in windows 4 or higher.) By Lemma 2, we can shift each of these cars toward window 1 as far as we want to. In order to retain the blocks of cars from windows 1 and 2 that were created in the optimal solution, we can shift the cars to fill window 2 while retaining the order in which they appear in the optimal solution. In doing so, we might displace some cars from window 3 that the optimal solution had put into window 2, and by taking cars that began in window 2 out of window 3 (by shifting them downward), we create holes in the window 3 solution. However, Lemma 1 tells us that we can slide the displaced cars upward into window 3 in the same manner, preserving their overall order as well, and we can slide the cars in window 3 upward to fill in any holes we have created.

In this manner, we have constructed a solution in which windows 1 and 2 contain only cars that began in those two windows, and windows 3 through $\frac{N}{W}$ contain only cars that began in those two windows. Furthermore, because we have preserved the order among cars in windows 1 and 2, and among cars in windows 3 and 4, the blocks of cars from windows k and $k+1$, for all odd k, in the optimal solution are preserved in this new feasible solution.

Now suppose that, for some odd number m, windows k and $k+1$ contain only cars that

were initially in windows k and $k+1$ and that all optimal blocks among those cars have been preserved, for all $1 \leq k \leq m$, k odd.

Consider window $m+2$. Ordinarily, cars from windows $m+1$, $m+2$, and $m+3$ could be assigned to window $m+2$. However, since all the cars from window $m+1$ are now in windows m or $m+1$, only cars from windows $m+2$ and $m+3$ can be in window $m+2$. So, as before, we need only look at window $m+3$. And again, Lemmas 1 and 2 ensure that we can slide cars that began in windows $m+2$ and $m+3$ downward into window $m+2$, and cars that began in windows $m+4$ and $m+5$ upward to fill in the holes in window $m+4$, while maintaining all optimal blocks of cars from windows $m+2$ and $m+3$, and from windows $m+4$ and $m+5$.

In this manner, we can construct a feasible solution where, for any odd k, windows k and $k+1$ contain only cars that began in windows k and $k+1$, and where all optimal blocks of cars from windows k and $k+1$ are preserved.

Since the two-by-two heuristic creates the best possible solution for each pair of windows k and $k+1$ (for odd k), it will do at least as well as any feasible solution, and thus will do at least as well as the solution we have constructed, which preserves all of the optimal blocks of cars from windows k and $k+1$. □

3.3.1 Even-k Postprocessing

Solutions created by the two-by-two heuristic account for possible interactions between blocks in consecutive windows k and $k+1$ for odd k, but do not account for similar interactions for even k. The heuristic optimizes windows 1 and 2 together and windows 3 and 4 together,

but not windows 2 and 3 although they are adjacent.

Even-k postprocessing takes the two-by-two heuristic solution as a starting point, and optimizes all even-k window pairs k and $k+1$. Notice that, since the two-by-two heuristic solution is the starting point, even-k postprocessing will never decrease the quality of the objective.

Two-By-Two Heuristic With Even-k Postprocessing

Begin with the pre-ordered solution

Run within-window preprocessing

For $k = 1$ to $\frac{N}{W} - 1$ do

    If k is odd

        Solve the optimization problem on the kth and $(k+1)$st window lengths

End for

For $k = 1$ to $\frac{N}{W} - 1$ do

    If k is even

        Solve the optimization problem on the kth and $(k+1)$st window lengths

End for

3.4 Incremental Window Algorithm

Given any feasible solution, we can improve the objective by selecting any part of the solution and locally optimizing. The within-window preprocessing algorithm does just this, over a one-window-length neighborhood. However, to achieve better solutions we would like to

enlarge the size of the neighborhood over which we optimize. Although the formulations we detailed in previous sections are still too large and slow-to-solve to optimality even with two-window-length neighborhoods, we can again use the within-window preprocessing technique before proceeding with our optimization.

Beginning from the initial preprocessed solution, we can optimize over the first two windows to obtain a new solution with an objective that is no worse than the objective of the starting solution. We can then "fix" the slot assignments of any paint blocks that begin in the first window, and optimize over the second and third windows (see Figure 3-2). If there are more than three windows, we can then fix the second window assignments, optimize over the third and fourth windows, and continue in this fashion until we have optimized locally over all windows.

Notice that after each optimization, it is possible that the solution generated might violate the order-within-color property. In fact, it might be advantageous to do so if, for some color c, the size of the c-block in window k is different from the size of the c-block in window $k+1$.

Thus, after each local optimization, we re-enforce the order-within-color property by swapping the cars in the violating blocks. The order-within-color property guarantees that we can do so without introducing any infeasibility in the solution; thus, we can only increase our flexibility by making this swap.

Incremental Window Algorithm

Begin with the pre-ordered solution

Run within-window preprocessing

For $k = 1$ to $\frac{N}{W} - 1$ do

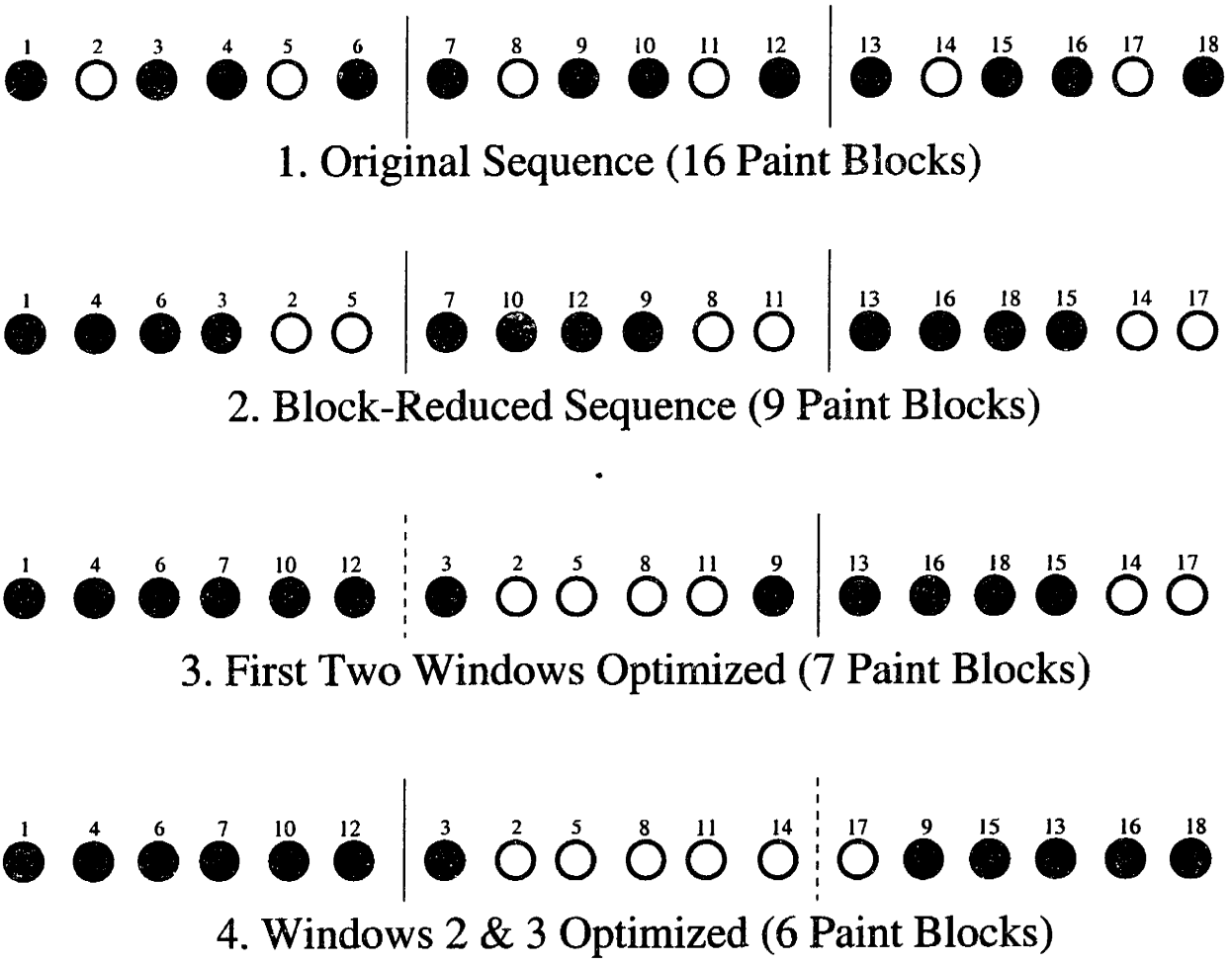

Figure 3-2: Incremental Window Algorithm

Solve the optimization problem on the kth and $(k+1)$st window lengths

Re-enforce the order-within-color property

Fix the positions of all cars whose blocks begin in the kth window length

End for

Symmetry might suggest that an equally good algorithm would be to begin at the end of the N cars and process the cars from the back to the front, rather than optimizing front-to-back. However, recall that a plant often removes cars from the line for repairs and later inserts them back into the line. In these situations, it is beneficial to give priority to optimizing the front of the line, and to optimize (locally) the back of the line as a function of the front, because the structure of the back of the line is likely to change before it enters the paint shop.

3.4.1 Lookahead Pricing

One deficiency of the Incremental Window Algorithm is that, when optimizing windows k and $k+1$, the algorithm does not account for the color blocks that appear in window $k+2$.

Frequently, windows $k+1$ and $k+2$, but not in window k, contain a block of some color c. When optimizing windows k and $k+1$, we would like our heuristic to place that block of color c in window $k+1$ rather than window k, if possible. If the algorithm places the block in window $k+1$, then when optimizing windows $k+1$ and $k+2$ it could possibly put the two blocks of color c together.

To facilitate these beneficial scenarios, we use lookahead pricing in our optimizations.

The primary objective is still to maximize the average paint block size in the two windows k and $k+1$; however, we also include terms in the objective function for blocks of color c beginning in window $k+1$ if window $k+2$ also contains a block of color c. By setting the objective coefficient of color c blocks in beginning in window $k+1$ to $\frac{1}{C+1}$, we ensure that putting color blocks together will always take priority; however, once the paint block size has been maximized in windows k and $k+1$, the IP solution will maximize the opportunities for putting blocks from windows $k+1$ and $k+2$ together in the next optimization.

Chapter 4

Worst-Case Bounds

In this chapter, we present worst-case bounds for some of our heuristic algorithms.

We will bound the heuristic solutions by counting the number of blocks that can be split by the heuristics and determining how many consecutive windows can contribute to a feasible color block. In most cases, we will implicitly use the result of Theorem 1, that any colors that are paired with respect to an odd value of k can remain paired in a two-by-two heuristic solution.

We will prove bounds between the two-by-two heuristic solution and any solution obtained after preprocessing. That is, the two-by-two heuristic solution will be compared with solutions where cars of the same color that begin in the same window must appear together in the reordering. We will show that the heuristic solution is bounded by a factor of $2\frac{1}{2}$ when no color is dominant, $3\frac{1}{2}$ when no window is monochromatic, and 5 in the general case. We will give examples that show all of these bounds to be tight.

We will also demonstrate a bound between the two-by-two heuristic solution and the

overall optimal solution. In particular, we will show that for any initial sequence of cars, some two-by-two heuristic solution contains no more than 3 times the number of blocks that are in the optimal reordering, and no two-by-two heuristic can contain more than 3 times the optimal number of blocks plus $\frac{N}{2W}$.

4.1 Two-By-Two Heuristic vs. Preprocessed Solution

In this section, our analysis will compare the heuristic solutions, all of which begin with the step of within-window preprocessing, with the optimal solution after within-window preprocessing.

4.1.1 Case 1: No Dominant Color

Definition and Probability

We define a color c to be *dominant* if $\frac{1}{3}$ or more of any window in the initial ordering is to be painted color c.

In the general case, we can calculate the chance of having no dominant color as follows. If p_c is the probability that any specific car is color c, the chance of that color c being $\frac{1}{3}$ or more of a given window-length is

$$\sum_{i=\lceil \frac{W}{3} \rceil}^{W} \frac{W!}{(W-i)!i!} p_c^i (1-p_c)^{W-i}.$$

Aggregating over all colors and all windows gives a final probability of

$$(\prod_{c=1}^{C} \sum_{i=\lceil \frac{W}{3} \rceil}^{W} \frac{W!}{(W-i)!i!} p_c^i (1-p_c)^{W-i})^{\frac{N}{W}}$$

of having some dominant color.

A randomly-generated problem instance (using our real-world size and frequency data) has a 99.9856% chance of having no dominant color.

Possible Block Combinations

Suppose windows k and $k+4$ both contained cars of color c in the original sequence. As shown in Figure 4-1, the cars from those two windows cannot be part of the same block in any feasible reordered solution.

The closest a car in window k could become to cars in window $k+4$ would be W slots higher, in window $k+1$. Similarly, the closest a car in window $k+4$ could become to the cars in window k would be W slots lower, in window $k+3$. Thus, at least an entire window, window $k+2$, would be between the cars. This window can contain only cars that started in windows $k+1$, $k+2$, and $k+3$. However, by assumption the initial sequence contains no dominant color; therefore, the number of cars of color c in these three windows must each be less than $\frac{W}{3}$. Thus windows $k+1$, $k+2$, and $k+3$ must contain less than W cars of color c, so they cannot bridge the gap between the cars originally in window k and the cars originally in window $k+4$ (see Figure 4-1).

Since cars that are initially in windows that are 4 apart cannot be part of the same block in a feasible reordering, neither can cars that are more than 4 windows apart. Therefore, the

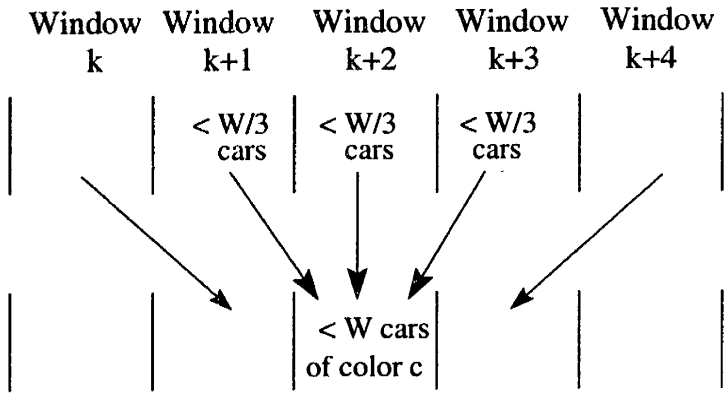

Figure 4-1: Windows k and $k+4$ cannot be in the same block

(k)	$(k, k+1, k+2)$
$(k, k+1)$	$(k, k+1, k+3)$
$(k, k+2)$	$(k, k+2, k+3)$
$(k, k+3)$	$(k, k+1, k+2, k+3)$

Table 4.1: Possible Block Combinations for No-Dominant-Color Case

window combinations shown in Table 4.1 are the only possible sets of windows whose cars (of the same color) can appear together in a block of a feasible solution/

Worst-Case Non-Dominant Bound for the Two-By-Two Heuristic

For each possible block combination, we can calculate the number of blocks into which the two-by-two heuristic will split it. There are two distinct cases: when k is odd, and when k is even. For example, when k is odd, the block composed of k and $k+1$ will fall into the same two-window span, so the heuristic will keep it a single block. On the other hand, when k is even, k will fall into one two-window span ($k-1$ and k) and $k+1$ will fall into the next two-window span ($k+1$ and $k+2$), so the heuristic will split it into two blocks. Notice that the two-by-two heuristic never splits a block into more than three pieces. Moreover, the

Combination	*Blocks (odd k)*	*Blocks (even k)*
(k)	1	1
$(k, k+1)$	1	2
$(k, k+2)$	2	2
$(k, k+3)$	2	2
$(k, k+1, k+2)$	2	2
$(k, k+1, k+3)$	2	3
$(k, k+2, k+3)$	2	3
$(k, k+1, k+2, k+3)$	2	3

Table 4.2: Two-By-Two Heuristic Results for Each Block Case

heuristic can split a block into three pieces only when k is even and k and $k+3$ are both part of the block.

Consider any block that contains cars from both k and $k+3$, for k even. Cars from window k can appear no higher than window $k+1$ in a feasible solution, and cars from window $k+3$ can appear no lower than window $k+2$. Therefore, the block must cross the boundary between windows $k+2$ and $k+3$. Because we are only looking at boundaries between an even-numbered window and the next (odd-numbered) window, there are only $\frac{N}{2W}$ such boundaries.

Therefore, the two-by-two heuristic will break an optimal solution containing B^* blocks into no more than $2B^* + \frac{N}{2W}$ pieces. Dividing through by the optimal number of blocks B^* gives a bound on the heuristic of $2 + \frac{N}{2WB^*}$.

To find the largest possible bound, we need to minimize B^*. Any sequence contains $\frac{N}{W}$ windows. Because the input sequence contains no dominant color, each window must contain at least 4 colors. Therefore, the initial sequence contains at least $\frac{4N}{W}$ blocks. We have shown that any feasible solution can combine at most 4 blocks. Consequently, any optimal solution

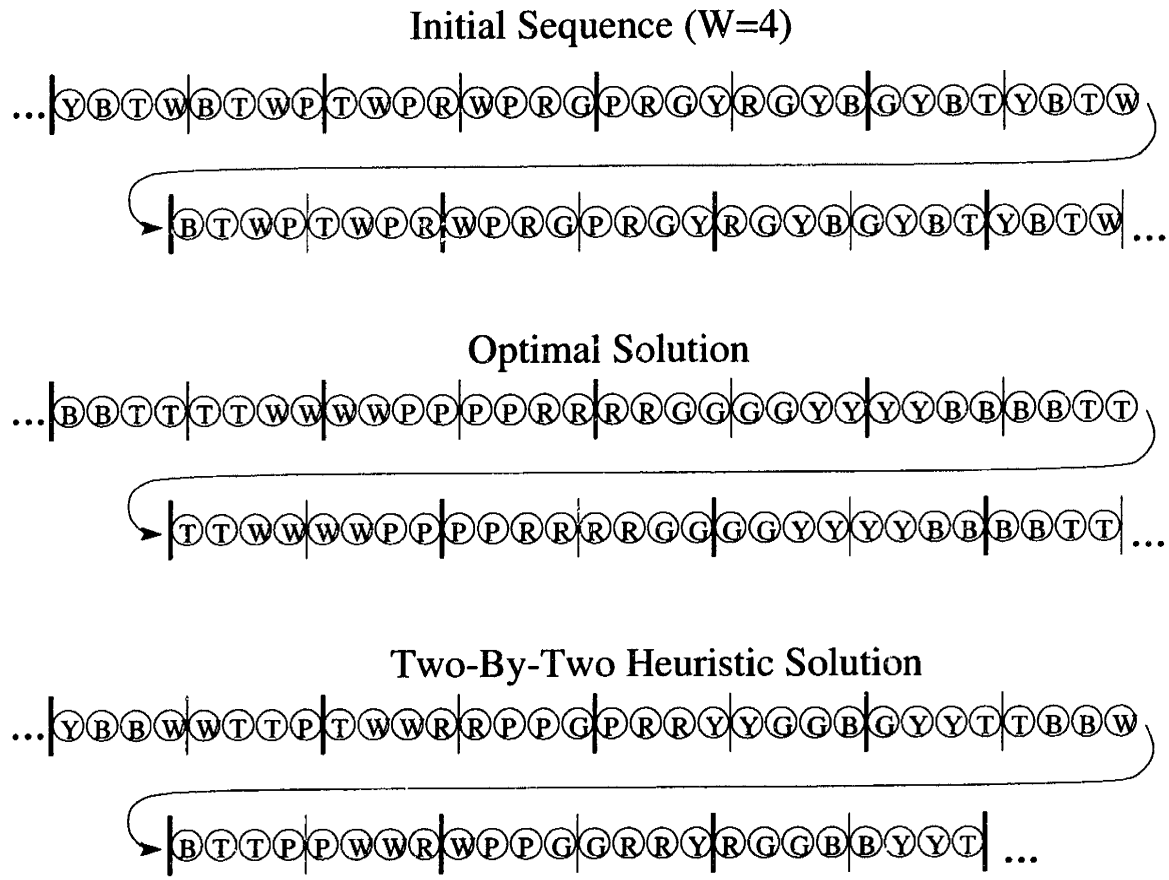

Figure 4-2: Tight Worst-Case Example for Initial Sequences With No Dominant Color

contains at least $\frac{4N}{4W} = \frac{N}{W}$ blocks, and thus $B^* \geq \frac{N}{W}$.

So, the bound on the two-by-two heuristic is $2 + \frac{N}{2W\frac{N}{W}} = 2\frac{1}{2}$. The example in Figure 4-2 shows that this bound is asymptotically tight.

The first two lines of Figure 4-2 show the initial (infinite) sequence of N cars. Each window of the initial sequence contains 4 cars, each of a different color, so no color is dominant. The optimal solution (shown in the next two lines of the figure) contains $\frac{N}{4}$ blocks. Each window boundary is the midpoint of a block containing one car from each of 4 windows. The two-by-two heuristic gives the solution shown in the final two lines of the figure, with a total of $\frac{5N}{8}$ blocks. The setup of the initial blocks causes each two-window set

to contain 5 colors, so the heuristic forms 5 blocks out of every two-window set.

Since $\frac{\frac{5N}{8}}{\frac{N}{4}} = 2\frac{1}{2}$, this example shows that our bound of $2\frac{1}{2}$ is tight for the two-by-two heuristic when the initial sequence contains no dominant color.

4.1.2 Case 2: No Monochromatic Window(s)

We say that a window of the initial sequence is "monochromatic" if every car in the window is the same color. Case 1, with no dominant color, is a special case of the non-monochromatic case.

Possible Block Combinations

As in Case 1 with no dominant color, we can consider what blocks any feasible solution might potentially contain.

Suppose some solution contained cars of color c from window k and window $k+6$ in the same block. The cars from window k could appear no higher than window $k+1$, and the cars from window $k+6$ could appear no lower than window $k+5$.

Consider window $k+3$. Because no window is monochromatic, this window must contain at least one car not of color c. By the window restrictions, we can assign this car to a slot in window $k+2$, window $k+3$, or window $k+4$. In all three of those cases, it would have to be between the cars of color c that began in window k and the cars of color c that began in window $k+6$ (see Figure 4-3). Therefore, the color c cars from windows k and $k+6$ could not be in the same block in the feasible solution.

Therefore, when analyzing the non-monochromatic case, we need to consider blocks con-

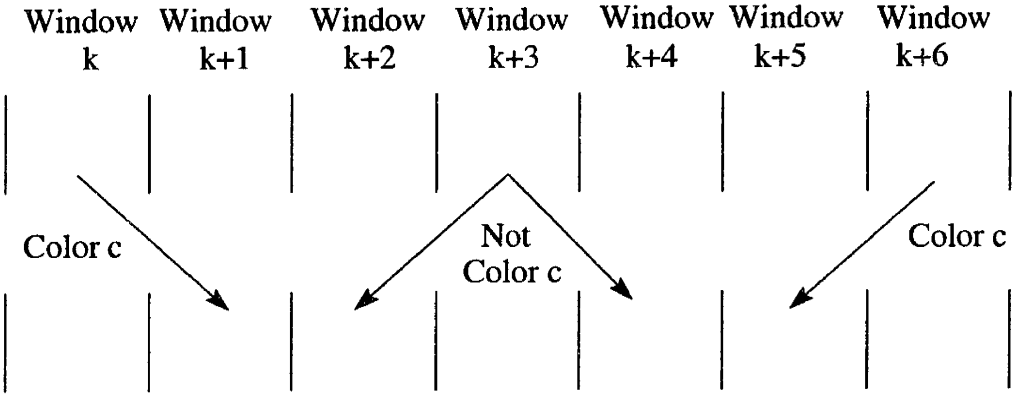

Figure 4-3: Windows k and $k+6$ cannot be in the same block

sisting only of cars from 6 consecutive windows, from window k to window $k+5$.

Worst-Case Non-Monochromatic Bound for the Two-By-Two Heuristic

Consider the case when k is odd. At most, a block in the optimal solution will have cars spanning from window k to window $k+5$ in the initial solution. The two-by-two heuristic will be able to group them into three blocks: windows k and $k+1$, $k+2$ and $k+3$, and $k+4$ and $k+5$. (Note that one or more of those might be empty, but there will be at most 3 blocks created from the one optimal block.)

When k is even, the two-by-two heuristic can create 4 blocks only when an optimal block contains cars from window k and window $k+5$. In that case, the heuristic will create blocks of cars from $k-1$ and k, $k+1$ and $k+2$, $k+3$ and $k+4$, and $k+5$ and $k+6$. (Windows $k-1$ and $k+6$, as well as some of the interior windows, do not actually need to have cars of color c.)

Consider any optimal block of color c that could be split into 4 blocks by the two-by-two heuristic. Because no window is monochromatic, window $k+2$ must contain a car that is not of color c. Let car j be the last such car in the initial sequence. Car j must either come

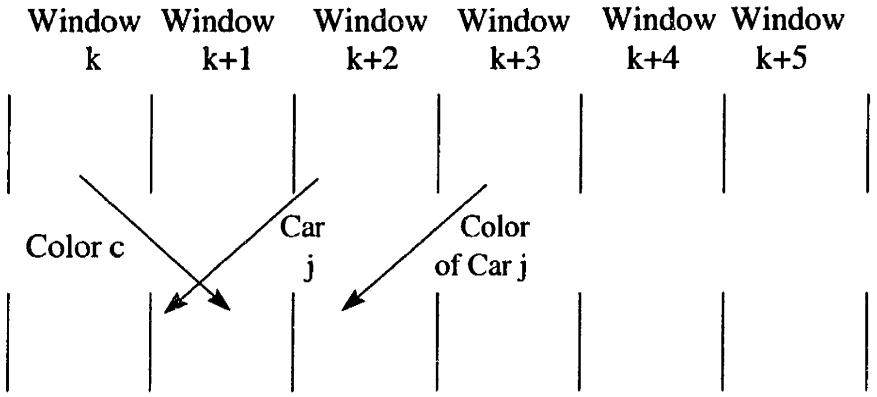

Figure 4-4: Car j must end a block

before or after the block of color c in the optimal solution. The highest slot car j could reach would be in window $k+3$. However, the car(s) of color c in window $k+5$ can reach down only to slots in window $k+4$. Therefore, car j cannot appear after the color c block in the optimal solution, and thus must appear before it (see Figure 4-4).

Similarly, consider any car not of color c in windows $k+3$ and higher. The lowest this car could reach is window $k+2$, but cars of color c in window k can reach up only to window $k+1$. Therefore, any car not of color c in windows $k+3$ or higher must appear after the color c block in the optimal solution.

So, car j (a car not of color c that is initially in window $k+2$) must appear before the block of color c, and (since j is the last such car in window $k+2$) every later car of the same color as j must appear after the block of color c. Therefore, by the order-within-color property, car j must be the last car of its block in the optimal solution.

Consider the block containing car j. From above, the last window that the block can contain cars from is window $k+2$. Since k is an even number, so is $k+2$. From our previous analysis, we know that the only way the two-by-two heuristic can create 4 blocks from a

single block of the optimal solution is for the optimal block to contain cars from windows l and $l+5$, where l is even. In that case, $l+5$ must be odd, so the block that is ended by car j will not be split into 4 by the two-by-two heuristic. Moreover, we have shown that any block of bound 4 must have such a car j, and thus will have such a "partner-block" that will not be split into 4. Therefore, at most one half of all blocks of the optimal solution will be split into 4 by the two-by-two heuristic; the rest will be split into at most 3.

Thus, we can bound the number of blocks in the two-by-two heuristic solution by a factor of $3\frac{1}{2}$ times the number of blocks in the optimal solution. Figure 4-5 demonstrates that this bound is asymptotically tight.

The first line of Figure 4-5 shows the initial (infinite) sequence. Each window contains two cars, each of a different color. Three consecutive windows contain green and red cars, the next three contain red and blue cars, and the next three contain blue and green cars. Thus each color appears in 6 consecutive windows. The second line of the figure shows the optimal solution, with a total of $\frac{N}{6}$ blocks. Each time a color appears (in 6 consecutive windows) its cars are consolidated into one block of size 6. The two-by-two heuristic can create the solution shown in the fourth line of Figure 4-5; this solution, as shown in the figure, will have $\frac{21N}{36}$ blocks. Every set of three consecutive 2-window sets will contain two sets containing 2 colors and one set containing 3 colors, for a total of 7 color blocks in every 6 windows, or 21 color blocks for every 36 cars. The ratio of the two solution values achieves the bound of $3\frac{1}{2}$.

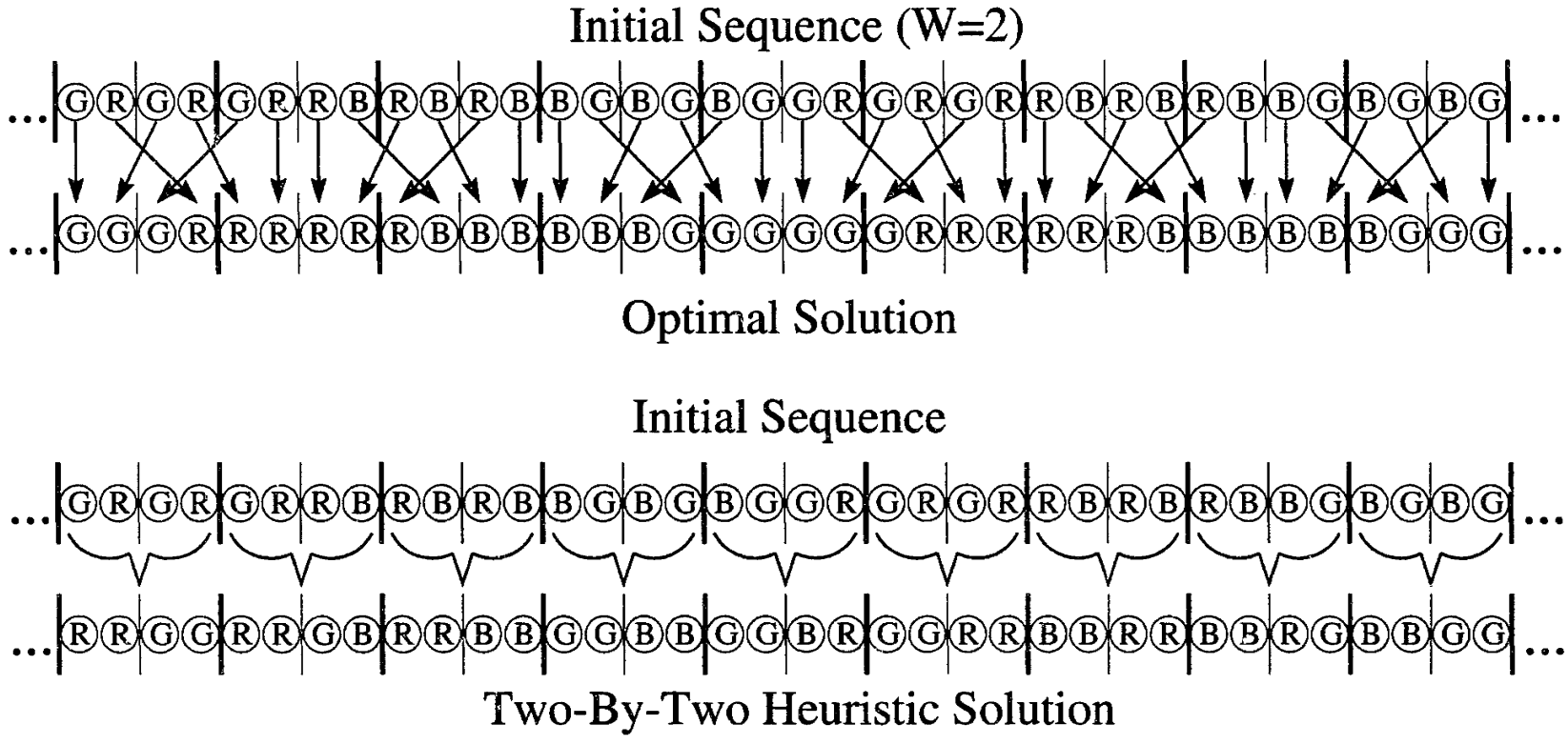

Figure 4-5: Tight Worst-Case Example For Non-Monochromatic Initial Sequences

4.1.3 Case 3: General Structure

In this section, we consider the most general initial sequences, permitting any window structure (including dominant colors and monochromatic windows).

Possible Block Combinations

In the proof of the non-monochromatic bound, we showed that a two-by-two heuristic solution will break any optimal block consisting of the cars from only 6 consecutive windows (k through $k+5$) into no more than 4 blocks. (Note that this part of the proof did not use the fact that windows were non-monochromatic, and thus is applicable to the general case as well.)

Now consider an optimal block of color c comprised of the cars from 7 or more consecutive windows k through l. The cars of color c from window k can reach no higher slots than those in window $k+1$, and the cars of color c from window l can reach no lower slots than those in window $l-1$.

Suppose some window m, $k+3 \leq m \leq l-3$, is non-monochromatic. Then m must contain a car j that is not color c. Car j can reach no lower than window $k+2$ and no higher than $l+2$, and so it can be placed neither before nor after the block of color c in the optimal solution (see Figure 4-6).

Therefore, the sequence can contain no such a car j; blocks $k+3$ through $l-3$ must all be monochromatic (and all must be color c).

Thus, any optimal block of color c comprised of cars from 7 or more consecutive windows must obey the following property: the windows whose cars form the block must include,

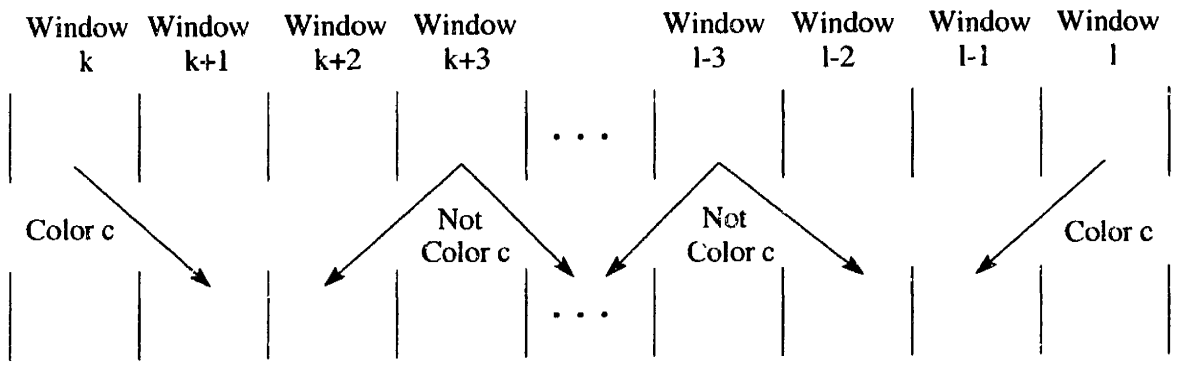

Figure 4-6: Windows $k+3$ through $l-3$ must be monochromatic

k	n	*Blocks*
odd	1	$(k,k+1)$; $(k+2,m_1)$; $(k+3,k+4)$; $(k+5,k+6)$
odd	2	$(k,k+1)$; $(k+2,m_1)$; $(m_2,k+3)$; $(k+4,k+5)$
odd	odd ≥ 3	$(k,k+1)$; $(k+2,m_1)$; $(m_2,m_3)...(m_{n-1},m_n)$; $(k+3,k+4)$; $(k+5,k+6)$
odd	even ≥ 4	$(k,k+1)$; $(k+2,m_1)$; $(m_2,m_3)...(m_{n-2},m_{n-1})$; $(m_n,k+3)$; $(k+4,k+5)$
even	1	$(k-1,k)$; $(k+1,k+2)$; $(m_1,k+3)$; $(k+4,k+5)$
even	2	$(k-1,k)$; $(k+1,k+2)$; (m_1,m_2); $(k+3,k+4)$; $(k+5,k+6)$
even	odd ≥ 3	$(k-1,k)$; $(k+1,k+2)$; $(m_1,m_2)...(m_{n-2},m_{n-1})$; $(m_n,k+3)$; $(k+4,k+5)$
even	even ≥ 4	$(k-1,k)$; $(k+1,k+2)$; $(m_1,m_2)...(m_{n-1},m_n)$; $(k+3,k+4)$; $(k+5,k+6)$

Table 4.3: Two-by-two heuristic blocks of size ≥ 7 for all cases of k and n

in order, (1) no more than 3 non-monochromatic windows k, $k+1$, and $k+2$; (2) any number n of monochromatic windows m_1 through m_n of color c; and (3) no more than 3 non-monochromatic windows $k+n+3$, $k+n+4$, and $k+n+5$.

General Worst-Case Bound for the Two-By-Two Heuristic

Table 4.3 shows the blocks that would be created by the two-by-two heuristic for every possible case of k and n. Consider the cases for which $n \geq 3$. The two-by-two heuristic creates one or more consecutive blocks that are all monochromatic in color c. Therefore, although the heuristic "counts" them as separate blocks, in the heuristic solution they comprise one

k	n	# of Blocks	Blocks
odd	1	4	$(k,k+1)$; $(k+2,m_1)$; $(k+3,k+4)$; $(k+5,k+6)$
odd	2	4	$(k,k+1)$; $(k+2,m_1)$; $(m_2,k+3)$; $(k+4,k+5)$
odd	odd ≥ 3	5	$(k,k+1)$; $(k+2,m_1)$; $(m_2...m_n)$; $(k+3,k+4)$; $(k+5,k+6)$
odd	even ≥ 4	5	$(k,k+1)$; $(k+2,m_1)$; $(m_2...m_{n-1})$; $(m_n,k+3)$; $(k+4,k+5)$
even	1	4	$(k-1,k)$; $(k+1,k+2)$; $(m_1,k+3)$; $(k+4,k+5)$
even	2	5	$(k-1,k)$; $(k+1,k+2)$; (m_1,m_2); $(k+3,k+4)$; $(k+5,k+6)$
even	odd ≥ 3	5	$(k-1,k)$; $(k+1,k+2)$; $(m_1...m_{n-1})$; $(m_n,k+3)$; $(k+4,k+5)$
even	even ≥ 4	5	$(k-1,k)$; $(k+1,k+2)$; $(m_1...m_n)$; $(k+3,k+4)$; $(k+5,k+6)$

Table 4.4: Number of two-by-two heuristic blocks of size ≥ 7 for all cases of k and n

single, large block of color c.

Table 4.4 therefore shows the number of real blocks into which the two-by-two heuristic can split an optimal block. Previously, we have shown that any optimal block of size less than 7 will be split into no more than 4 blocks by the two-by-two heuristic. Therefore, since no optimal block will be split into more than 5 blocks by the two-by-two heuristic, the two-by-two heuristic solution can be bounded by a factor of 5 times the optimal solution.

The example of Figure 4-7 demonstrates that the bound is tight.

The top line of Figure 4-7 shows an (infinite) initial sequence of cars. The sequence alternates 3 windows with 2 colors each (blue and red) with 3 monochromatic windows (either blue or red) so that the optimal solution is comprised entirely of blocks containing cars from 9 consecutive windows, the interior 3 of which are monochromatic. The second line shows the optimal solution, in which each block contains 12 cars so that the total number of blocks is $\frac{N}{12}$. The fourth line shows a possible two-by-two heuristic solution that contains 10 blocks for every 24 cars, for a total of $\frac{10N}{24}$ blocks. The ratio of the number of heuristic blocks to the number of optimal blocks is 5, so the bound is indeed tight.

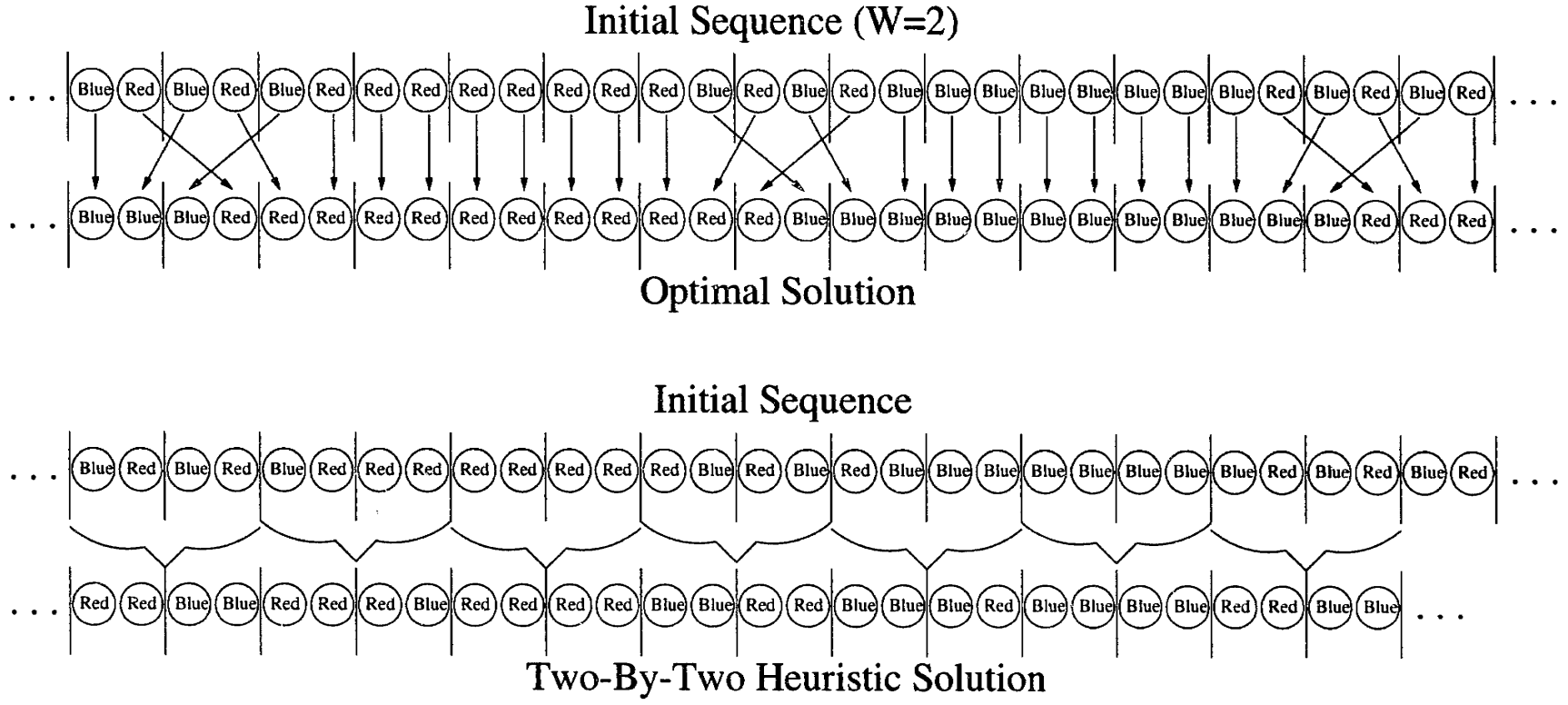

Figure 4-7: Tight Worst-Case Example For General Initial Sequences

4.2 Two-By-Two Heuristic vs. Optimal Solution

In this section, we prove a performance bound on the two-by-two heuristic with respect to the optimal (non-preprocessed) solution.

As before, we use the ranges of possible windows for any car as a key element of the proof. Consider the cars that the optimal solution assigns to window 1. The window restrictions imply that these cars must have started in windows 1 and 2. The cars in window 2 of the optimal solution might have started in windows 1, 2, or 3. If they started in window 3, they must come after all cars of the same color from windows 1 and 2 (by the order-within-color property). We can then split this color block, moving the window 3 cars into the third window and exchanging them for window 2 cars that were assigned to the third window of the optimal solution. By Lemmas 1 and 2, as well as Theorem 1, we know that we can exchange these cars and maintain feasibility.

We can repeat this process for all odd values of k between 1 and $\frac{N}{W}$ to obtain a feasible solution in which, for any odd value of k, windows k and $k+1$ contain only cars that were in windows k and $k+1$ in the initial solution.

Note that this procedure will split each color block in the optimal solution into at most three blocks: the beginning and end of each block might have been split off, but the rest remains intact. Thus, we can guarantee that some two-by-two heuristic solution contains no more than three times the optimal number of blocks exists.

However, we cannot guarantee that the two-by-two heuristic will find such a solution. The process described above takes advantage of the possibility that blocks might cross even-k boundaries (boundaries between windows k and $k+1$ for even values of k). The two-by-two

heuristic, as we have described it, does not look for these possibilities and, therefore might construct a worse solution. Specifically, for each even-k boundary, the two-by-two heuristic might miss an opportunity to reduce the number of color blocks by one. Therefore, we can bound the number of color blocks in any two-by-two heuristic solution by three times the number of blocks in the optimal solution, plus $\frac{N}{2W}$.

Figure 4-8 repeats the example of Figure 4-7, but shows the best possible two-by-two heuristic solution, which gives a tight bound of 3. The two-by-two heuristic solution shown in Figure 4-7 has a bound of $3 + (\frac{2}{3})(\frac{N}{2W})$.

In Figure 4-8, the algorithm has assigned the blue car in window 3 to the first slot in window 3 and enabled it to become part of the blue block in window 2. In Figure 4-8, the same blue car is assigned to a slot in window 4. In addition to breaking the color block, the assignment of the blue car to window 4 guarantees that the even-k postprocessing procedure cannot put the blue cars together.

The assignments shown in Figures 4-7 and 4-8 for windows 3 and 4 and windows 9 and 10 contain the best possible two-by-two heuristic solution (with 3 times the optimal number of blocks) and the worst possible two-by-two heuristic solution (with 5 times the optimal number of blocks). The two-by-two heuristic cannot determine *a priori* which assignment would be better.

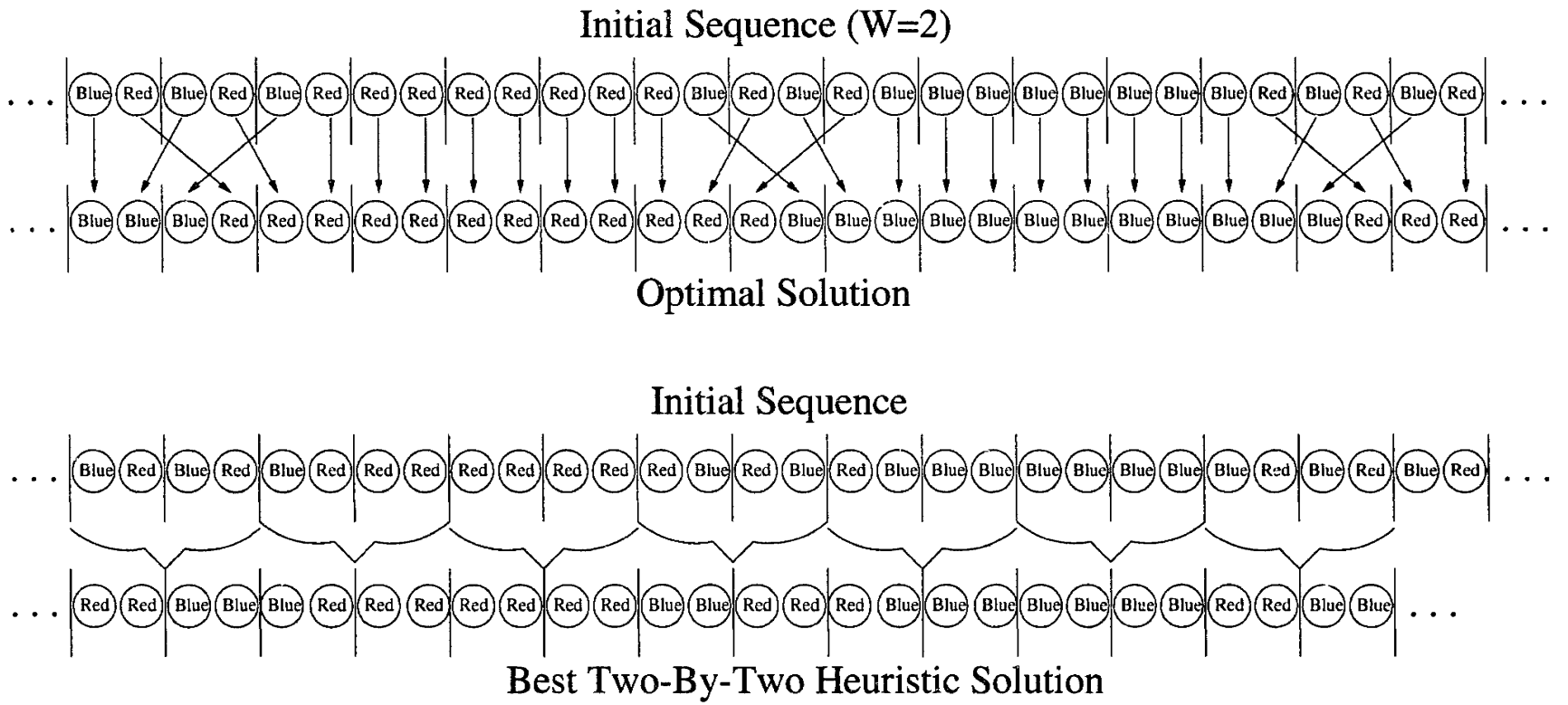

Figure 4-8: Tight Worst-Case Example of the Best Two-By-Two Solution

4.3 Two-By-Two Heuristic With Even-k Postprocessing

Because the two-by-two heuristic might miss the possibility of combining a color across an even-k boundary, even-k postprocessing might increase the quality of a two-by-two heuristic solution. The two-by-two heuristic with even-k postprocessing uses the two-by-two heuristic solution as a starting point, so it cannot do any worse than the two-by-two heuristic. However, it might not be able to do any better. For example, in Figures 4-5 and 4-7, adding the even-k postprocessing will not have any effect.

4.4 Incremental Window Heuristic

As demonstrated by the computational results shown in Chapter 6, the incremental window heuristic has the potential to do much better than the two-by-two heuristic (with or without even-k postprocessing). For the examples shown in Figures 4-2, 4-5, and 4-7, the incremental window heuristic can achieve the optimal solution.

However, it is also possible that the incremental window heuristic will do worse than the two-by-two heuristic.

The top line of Figure 4-9 shows an initial sequence of 12 cars. The second line shows a solution obtainable using the two-by-two heuristic. This solution groups the red cars from window 2 with the red car from window 4 to obtain an optimal solution (5 blocks). The third through fifth lines show the same sequence after one, two, and three stages of the incremental window heuristic. The incremental window heuristic's lookahead abilities reap

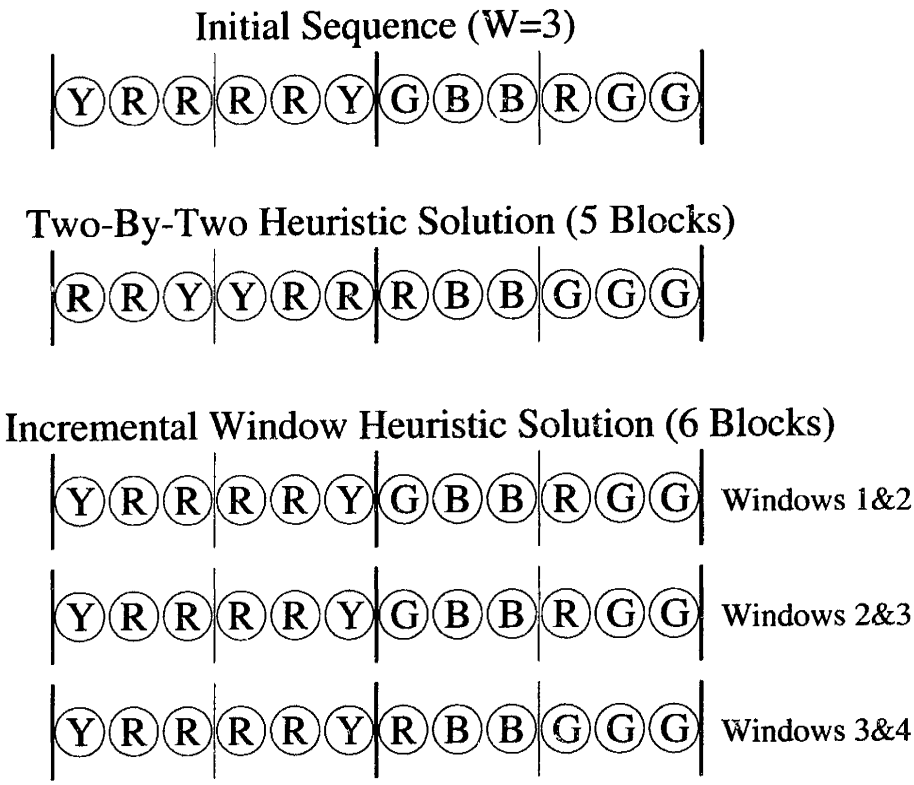

Figure 4-9: Example in which the two-by-two heuristic beats the incremental window heuristic

no benefit because the next red car is in window 4, not window 3, and the heuristic solution is suboptimal (6 blocks). The heuristic has no incentive to make any changes in the first two stages (windows 1 and 2, and windows 2 and 3), and can group the green cars only in the final stage (windows 3 and 4).

4.5 Summary of Bounds

We can summarize the results of this chapter with the following theorem.

Theorem 2 *For any initial sequence of cars, (i) there exists a two-by-two heuristic solution that contains no more than 3 times the optimal number of paint blocks, and no two-by-two heuristic solution will contain more than 3 times the optimal number of paint blocks plus* $\frac{N}{2W}$*; (ii) no two-by-two heuristic solution will have more than* f *times the number of paint blocks required by the preprocessed optimal solution, where* $f = 2\frac{1}{2}$ *if no color is dominant,* $f = 3\frac{1}{2}$ *if no window is monochromatic, and* $f = 5$ *in the general case; (iii) even-k postprocessing may not improve the two-by-two heuristic solution, even in the worst cases; and (iv) the incremental window algorithm can give solutions that are either better or worse than the two-by-two heuristic.*

The proofs of all four parts of Theorem 2 are given in previous sections of this chapter.

Chapter 5

Color-Dependent Cost Structure

In creating our models, designing our heuristics, and analyzing bounds on algorithms, we have assumed that each color change has the same cost. In reality, the cost of purging different colored paints from the lines might depend on the color; for example, black is generally a harder color to purge than white. The real-world data that we were given shows that the 15 colors range in cleanup costs from \$10 per purge to \$31 per purge.

We can demonstrate that all of the properties, models, solution algorithms, and bounds that we have given for the 0/1-cost problem have analogs for the color-dependent-cost problem.

5.1 Order-Within-Color for the Case With Color-Dependent Costs

Our proof of the order-within-color property for the 0/1-cost problem was based on the fact that for every feasible reordering of the cars, some feasible order-within-color solution had the same color car in each slot. That is, for every slot j, if car i was in the slot in some feasible reordering, then slot j would contain a car of color c_i (though not necessarily the same one) in the corresponding order-within-color solution. We also used the fact that the order-within-color solution would have the same cost as the original reordering.

Both facts remain valid when the costs are color-dependent. The first fact concerns feasibility, and is independent of costs. Therefore, it will certainly be true in the color-dependent-cost case. The second fact, that the costs of the initial reordering and the order-within-color solution are the same, remains valid because (i) every slot in the order-within-color solution contains a car of the same color as in the feasible reordering, and (ii) since the costs are color-dependent, the solutions have the same cost because they have the exact same color changes. Therefore, the order-within-color property holds for the color-dependent-cost problem.

This fact will remain valid even if the costs were generalized even further, to situations when the cost was dependent on both the color being cleaned out and the next color being painted.

5.2 Formulations

5.2.1 Effects on IP Formulations with Strong Relaxations

By making a small change in the objective functions of our formulations, we can easily account for this more generalized cost structure. Suppose we define v_c to be the cost of purging paint of color c, and define c_i to be the paint color of car i.

Recall that if I_{ij} is a 0/1 indicator variable that is 1 if and only if car i is in slot j and the next car of color c_i is in slot $j+1$, then the objective function for the 0/1 order-within-color disaggregated savings indicator variable ($ODSIF$) formulation is

$$\text{maximize} \sum_{i,j} I_{ij}.$$

For the more general case, we have saved one cleaning of paint color c_i, so in the color-dependent case, we can modify the objective to be

$$\text{maximize} \sum_{i,j} v_{c_i} I_{ij}.$$

Now the objective weights each color change by the cost to purge that color paint from the lines.

We can make a similar adjustment to the objective function of the 3-dimensional shortest-path-based formulation ($OSP3$). In the 0/1-cost case, the objective is

$$\text{maximize} \sum_{j=\text{cnext}(i)} x_{ijt}.$$

In this formulation, the 0/1 variables x_{ijt} are 1 if and only if car i is in slot t and car j is in slot $t+1$, and cnext(i) is the next car of color c_i.

As before, we can model the color-dependent costs by changing the objective coefficients:

$$\text{maximize} \sum_{j=\text{cnext}(i)} v_{c_i} x_{ijt}.$$

We can change the complete enumerative formulation (CEF) in the same way. The objective function for the 0/1-cost case is

$$\text{maximize} \sum_{b \in B} \sum_{j: l_b \le j \le u_b} (s_b - 1) x_{bj}.$$

In this model, x_{bj} is a 0/1 variable set to 1 if and only if block b is chosen and begins in slot j, B is the set of all potential blocks, s_b is the size of block b, and l_b and u_b are the upper and lower slot boundaries for the start of block b.

This objective function counts each transition between cars of the same color within each block; a block of size s_b will have $s_b - 1$ such transitions. As in the other formulations, every transition between cars of the same color c will now save v_c (rather than 1), so our objective function for the color-dependent-cost version of the CEF becomes

$$\text{maximize} \sum_{b \in B} \sum_{j: l_b \le j \le u_b} v_{c_b} (s_b - 1) x_{bj}.$$

5.2.2 Effect on Dynamic Programming Formulation

Recall that in our dynamic programming formulation, we defined the cost-to-go as $J_n(c, n_1, n_2, \ldots, n_{C-1})$ for the state in which (i) the first n slots have been filled by n_1 cars of color 1, n_2 cars of color 2, ..., n_{C-1} cars of color $C-1$, and $n - \sum_{i=1}^{C-1} n_i$ cars of color C, and (ii) the last slot filled contains a car of color c. $J_n(c, n_1, n_2, \ldots, n_{C-1}) = \min\{J_{n-1}(1, n_1 - 1, n_2, \ldots, n_{C-1}), J_{n-1}(2, n_1, n_2-1, \ldots, n_{C-1}), \ldots, J_{n-1}(C-1, n_1, n_2, \ldots, n_{C-1}-1), J_{n-1}(C, n_1, n_2, \ldots, n_{C-1})\}$.

Because the order-within-color property still holds for when costs are color-dependent, specifying the number of cars of each color that have been used already still uniquely defines the exact set of cars that are available, so the dynamic programming formulation remains valid for the case in which costs are color-dependent.

5.3 Upper Bounds

5.3.1 Upper Bound Calculation

Both of the specialized methods we presented for generating upper bounds for 0/1-cost problems are valid for color-dependent cost problems as well.

Recall that the time-relaxed version of the complete enumerative formulation, which we solve using a greedy algorithm to provide an upper bound on the objective of the CEF, decomposes by color. Therefore, since each color is optimized separately, color-dependent costs will have no effect on the validity of the process; the heuristic will minimize the number of blocks created of each color.

For the three-dimensional shortest-path based formulation $OSP3$, the formulation of the 0/1-cost problem and of the color-dependent cost problem are the same, except for the objective function. Therefore, the underlying structure is still an acyclic shortest-path problem with one side constraint for each car, so our Lagrangean relaxation procedure is still valid.

Further, our calculation speedup will also still work. In the 0/1-cost problem, we exploit the fact that, excluding the benefit of having two same-colored cars in a row, the cost of any edge from car i in slot t to car j in slot $t+1$ is the same as the cost of the edge from car i in slot t to car k in slot $t+1$. This is true because the Lagrange multipliers on the side constraints depend only on car i, not car j (or k) or the slot.

In the color-dependent-cost problem, this fact remains valid. Ignoring the benefit of having two identically-colored cars in a row (and therefore assuming that every edge constitutes a color change), the edge costs will still be the same; the cost of the edge from car i in slot t to car j in slot $t+1$, excluding the same-color benefit (if any), is just the Lagrange multiplier μ_i for car i plus the cost v_{c_i} of cleaning color i. The edge costs, then, are still independent of the slot t and the incoming car j.

Therefore, our method of comparing, for every car j, the minimum among eligible predecessor cars i of the edge costs ($\mu_i + v_{c_i}$ in this case) with the edge cost of the previous car of color c_j (without the cleanup cost v_{c_i} in this case) is still valid and will still give us the same speedup in our calculations.

Note however that we lose this property when the transition cost depends upon the colors of two successive cars.

5.3.2 Upper Bound Equivalence

In the 0/1-cost case, we were able to show that all three of our strong formulations ($ODSIF$, $OSP3$, and CEF), although they have different feasible sets, will always generate the same upper bound. We now show that this property also applies for situations with color-dependent costs.

In adding color-dependent costs to our model, we have not altered any of the constraints. Therefore, the correspondences we showed between feasible solutions to the various formulations remain valid.

Comparison of $OSP3$ and $ODSIF$

We showed that any feasible solution $\hat{x}$ to the relaxation of $OSP3$ has a corresponding feasible solution $(\bar{x}, \bar{I})$ to the relaxation of $ODSIF$ with a one-to-one correspondence between the values of $\hat{x}_{i(cnext(i))t}$ and $\bar{I}_{it}$, satisfying the conditions $\bar{I}_{it} = \hat{x}_{i(cnext(i))t}$ and $\bar{x}_{it} = \sum_j \hat{x}_{ijt}$.

The coefficient of $\hat{x}_{ijt}$ in the color-dependent-cost $OPS3$ formulation is v_{c_i} if $j = \text{cnext}(i)$ and zero otherwise. The coefficient of $\bar{x}_{it}$ in the color-dependent-cost $ODSIF$ formulation is zero and the coefficient of $\bar{I}_{it} = v_{c_i}$. Therefore, the two solutions will have the same objective value.

On the other hand, we showed that every optimal solution to the relaxation of $ODSIF$ must be I-maximal. As long as the paint-cleaning costs are positive (which is a reasonable assumption), the same proof applies to the color-dependent-cost case. We also showed that every I-maximal solution $(\bar{x}, \bar{I})$ to the relaxation of $ODSIF$ has a corresponding feasible solution to the relaxation of $OSP3$ in which $\hat{x}_{i(cnext(i))t} = \bar{I}_{it}$, so as above, the two will have

the same objective value. Because this conclusion is based only on feasibility, it applies to the color-dependent-cost case.

Therefore, when costs are color-dependent, we have shown that any optimal solution to the relaxation of $OSP3$ has a corresponding feasible solution to the relaxation of $ODSIF$ with the same objective value, and vice versa, so that the two relaxations will have the same optimal objective value.

Comparison of $OSP3$ and CEF

Using flow decomposition techniques, we showed that for any solution $\hat{x}$ to the relaxation of $OSP3$, we can create a feasible solution $\tilde{x}$ to CEF so that for each path p into which we can decompose the flow $\hat{x}$, $\sum_{b,t\in p}(s_b - 1)\tilde{x}^p_{bt} = \sum_{i,t\in p}\hat{x}^p_{i(cnext(i))t}$.

Since the objective value of $\hat{x}$ equals $\sum_{i,t} v_{c_i}\hat{x}_{i(cnext(i))t}$ and the objective value of $\tilde{x}$ equals $\sum_p \sum_{b,t\in p} v_{c_b}(s_b - 1)\tilde{x}^p_{bt}$, the objectives are equal:

$$\sum_p \sum_{b,t\in p} v_{c_b}(s_b - 1)\tilde{x}^p_{bt} = \sum_p \sum_{i,t\in p} v_{c_i}\hat{x}^p_{i(cnext(i))t} = \sum_{i,t} v_{c_i}\hat{x}_{i(cnext(i))t}.$$

On the other hand, we showed that every optimal solution to the relaxation of CEF must be $\tilde{x}$-maximal. Again, as long as the paint-cleaning costs are positive, the same proof applies to the color-dependent-cost case. We have shown that every $\tilde{x}$-maximal solution $\tilde{x}$ to the relaxation of CEF has a corresponding feasible solution to the relaxation of $OSP3$ in which $\sum_{b,t\in p}(s_b - 1)\tilde{x}^p_{bt} = \sum_{i,t\in p}\hat{x}^p_{i(cnext(i))t}$. As before, then the two relaxations have the same objective value.

So, we have now shown that in the case of color-dependent costs, the objective values of

the relaxations of $OSP3$ and $ODSIF$ will be equal and the objective values of the relaxations of $OSP3$ and CEF will be equal, and therefore the relaxations of all three formulations will have the same optimal objective value and thus provide the same upper bound.

5.4 Algorithms and Bounds

None of the heuristic algorithms we presented for the 0/1-cost problem use the fact that costs are all either zero or one. Therefore, they are all valid for the color-dependent-cost problem. In the Computational Results section, we will see that the empirical performance of the heuristics is approximately the same on the two cost structures.

Furthermore, all of the arguments we used in proving algorithmic bounds examined cars of a single color. Therefore, just as in the case of the greedy algorithm, color-dependent costs will not affect any of our proofs, and the bounds for the 0/1-cost case remain valid for the problem with color-dependent costs.

Chapter 6

Computational Results

We tested our heuristics on randomly-generated problems based on real-world data. Using the color frequencies provided in the Ford data and shown in Table 6.1 [22], and using the reasonable assumption that the color of each car was independent and identically distributed (iid), we generated sequences of 50 to 1500 cars and solved each using window sizes ranging from 1 to the minimum of 150 and $\frac{N}{10}$. We generated and solved ten instances of each case.

We conducted the computations on a Pentium II 400 MHz PC with 384 MB of RAM running on a Linux operating system. We used CPLEX 6.0 to solve all of the linear and integer programs, with the exception of the greedy upper bounding procedure for *TREL* and the shortest-path Langrangean subproblems for the linear programming relaxation of *OSP*3.

The results of our tests (see Table 6.2) showed that the length N of the sequence, as long as it is sufficiently longer than the window size W, does not have an effect on the quality of the solutions generated by any of our heuristics.

Color	*Frequency*
1	0.040
2	0.135
3	0.100
4	0.025
5	0.015
6	0.030
7	0.050
8	0.010
9	0.075
10	0.040
11	0.125
12	0.095
13	0.075
14	0.125
15	0.060

Table 6.1: Color Frequencies

The four heuristics (two-by-two, two-by-two with even-k postprocessing, incremental window, and incremental window with lookahead pricing) performed similarly. However, for every problem size the incremental window heuristic with lookahead pricing performed slightly better than the other heuristics. As shown in Table 6.2, our heuristic methods performed well for large window sizes, but poorly for small window sizes. However, for small window sizes we are able to solve the preprocessed block enumerative formulation exactly using integer programming branch-and-bound within a reasonable amount of time (no more than 10 minutes). Therefore, when the heuristic performs poorly, we can use an alternative method to obtain a good solution. When the exact method becomes too large (for window sizes of 25 and up), the heuristic performs well.

For the specific real-world case of $N = 750$ cars and a window size of $W = 75$ slots in either direction, the incremental window heuristic with lookahead pricing was, on average,

Window Size	*Two-By-Two*	*Two-By-Two With Even-k Postprocessing*	*Incremental Window*	*Incremental Window With Lookahead Pricing*
1	79.7%	79.7%	79.7%	70.1%
5	37.8%	30.7%	32.4%	24.0%
10	22.5%	16.4%	21.9%	14.4%
15	13.7%	10.0%	14.8%	9.5%
25	9.2%	8.1%	9.4%	6.2%
50	5.6%	4.0%	4.2%	3.2%
75	3.4%	2.2%	2.4%	2.0%
100	3.3%	2.2%	2.3%	2.0%
150	2.1%	1.5%	1.6%	1.5%

Table 6.2: Percent Gaps Between Upper Bounds and Heuristic Solutions for 0/1 Cost Structure

2.0% from the upper bound and solved in an average of 103 seconds (see Table 6.3). The solution times ranged from 44 seconds to 306 seconds (with only one instance requiring more than 2 minutes) and the solution gaps ranged from 1.5% to 2.5%. We also tested our heuristics on the color-dependent cost data provided with our real-world instance (see Table 6.4). The results were quite similar, and for the large-window cases which require a heuristic solution, these gaps were even smaller than in the 0/1-cost case.

Trial	*Initial Value*	*Final Value*	*Upper Bound*	*Percent Gap*	*Time (Seconds))*
1	60	671	681	1.5	75
2	78	667	684	2.5	306
3	68	667	680	1.9	49
4	92	671	685	2.0	98
5	65	667	680	1.9	78
6	68	666	682	2.3	44
7	72	670	686	2.3	78
8	74	668	680	1.8	92
9	59	667	682	2.2	87
10	62	668	681	1.9	120
Avg.	70	668	682	2.0	103

Table 6.3: Incremental Window With Lookahead Pricing Results For Real-World Cases (N = 750, W = 75)

Window Size	*Two-By-Two*	*Two-By-Two With Even-k Postprocessing*	*Incremental Window*	*Incremental Window With Lookahead Pricing*
1	79.9%	79.9%	79.9%	70.4%
5	38.0%	30.8%	32.6%	23.3%
10	22.8%	16.7%	21.4%	14.1%
15	14.4%	10.0%	14.7%	9.0%
25	9.3%	6.7%	9.4%	6.0%
50	5.4%	3.7%	4.1%	3.1%
75	3.2%	1.9%	2.4%	1.9%
100	2.9%	1.9%	2.2%	1.9%
150	1.9%	1.3%	1.5%	1.2%

Table 6.4: Percent Gaps Between Upper Bounds and Heuristic Solutions for Color-Dependent Cost Structure

Chapter 7

Summary

We have studied paint blocking, an important issue that arises in the automobile manufacturing industry. Because automobile manufacturing is an assembly-line industry, manufacturers must sequence the cars that they produce each day. Their sequencing criteria generally focus on load-balancing, with little or no emphasis on the color each car is to be painted. However, each time a plant paints two cars of different colors consecutively, it must clean ("purge") the old paint from the lines and insert the new paint before it can process the next car through the paint shop. Paint purges cost, on average, about \$15 - \$20, ranging from \$10 - \$30 depending on the color of the paint to be purged.

We began by modeling the problem as a traveling salesman problem with time windows (TSPTW). The TSPTW is an NP-hard problem, and the best solution algorithms from the literature are able to solve instances of up to 200 cities and a 40-slot window size, with solution times measured in hours. Our real-world problem has 750 cars (cities), each with a 151-slot window. We need to be able to generate solutions within a few minutes because it is

often necessary to reoptimize at various times during the day when the initial sequence has been disrupted, for example, by unscheduled rework to one or more of the cars, and it would not be feasible to shut down the production line for an hour until the system calculated a new solution.

Because the problem is so large and hard, we attempted to solve it using heuristic solution algorithms, combined with an upper-bounding procedure that would provide guarantees on the solution quality. We formulated a number of integer programming models for the problem, but discovered that their linear programming relaxations provided very weak upper bounds.

We proved an *order-within-color* property, and found that adding it to two of our best formulations, a disaggregated savings indicator formulation ($DSIF$) and a 3-dimensional shortest-path-based formulation ($SP3$), dramatically tightened the bounds, often to the exact optimal value of the integer program. The order-within-color property also enabled us to create a third useful formulation, the complete enumerative formulation (CEF), that enumerates all of the possible color blocks. Without the order-within-color property, this formulation would contain $O(10^{150})$ such color blocks in a real-world-sized problem, and be highly impractical; however, using order-within-color drastically reduces the number of color blocks, to no more than $11,000$ for any of our test cases. The relaxation of the CEF also gives a very tight upper bound, and we were able to prove that, although the relaxations of $ODSIF$, $OSP3$, and CEF have different feasible regions, they will always give the same upper bound on the integer optimal solution.

One final implication of the order-within-color property is that we were able to define an

$O(N^C)$ dynamic programming formulation for the problem. Although not helpful for our real-world problem with 15 colors, this formulation could be very useful for situations with less colors; for example, using this formulation would permit us to easily exactly optimize a limousine production line with no more than 2 or 3 colors.

Although the relaxations for three formulations whose provide the (same) good upper bound on the value of the optimal integer programming objective value, none of the three are easy to solve. In fact, we can solve only one of these models ($ODSIF$) directly, and even that takes 20 minutes to generate a bound. However, using a Lagrangean relaxation procedure and exploiting the fact that multipliers are only car-dependent permits us to generate a bound from the $OSP3$ formulation in approximately 8 minutes. Also, by eliminating the time constraints of the CEF model, we create a time-relaxed formulation $TREL$ that is solvable in under one second using a greedy algorithm. This algorithm gives us an upper bound on the LP bound, which is often exact, and (empirically) always relatively tight.

To generate good feasible solutions to the problem, we created heuristic methods: a two-by-two heuristic (with and without even-k postprocessing) and an incremental window heuristic (with and without lookahead pricing). All four heuristics began with a within-window preprocessing stage. Although the four heuristic solution methods all generated solutions with similar numbers of paint blocks, the incremental window heuristic with lookahead pricing consistently gave the best performance. For real-world problems, the incremental window heuristic with lookahead pricing averaged solutions that were 2.0% from the upper bound.

We were able to prove some performance bounds on the two-by-two heuristic (with and

without postprocessing) compared with the optimal solution to the preprocessed problem. In general, the number of blocks in the solutions generated are no more than 5 times the number of blocks in the optimal solution, but that number drops to $3\frac{1}{2}$ if no window is monochromatic and $2\frac{1}{2}$ if the input sequence contains no dominant color. Problem instances that are randomly generated to conform to the specifications of our real-world data are 99.9856% likely to have no dominant color.

Finally, we showed that all of the models, algorithms, properties, bounds, and proofs that were valid for the 0/1-cost case (in which every color change costs the same amount) remain valid for the color-dependent-cost case (in which the cost of purging paint depends on the color being purged). In empirical tests, our algorithms performed approximately as well on color-dependent problems as they did on 0/1 problems, and for real-world data they achieved slightly better results on color-dependent problems than on 0/1 problems.

Bibliography

[1] Adenso-Diaz, B., M. Gonzalez, and E. Garcia. "A Hierarchical Approach to Managing Dairy Routing." *Interfaces* 28 (2), pp. 21-31, 1998.

[2] Ahuja, R., T. L. Magnanti, and J. B. Orlin. *Network Flows.* 1993.

[3] Ascheuer, N., M. Fischetti, and M. Grotschel. "A Polyhedral Study of the Asymmetric Traveling Salesman Problem With Time Windows." Preprint SC 91-11, Konrad-Zuse-Zentrum fur Informationstechnik, 1997.

[4] Atassi, F. R. *Implementation of Block Painting in Ford's In-Line Vehicle Sequencing Environment.* MIT Master's Thesis, Department of Aeronautics and Astronautics, 1996.

[5] Balas, E. and A. S. Schulz. "The Traveling Salesman Problem With Time Windows." Working Paper, July 1995.

[6] Bertsimas, D. J. and J. N. Tsitsiklis. *Introduction to Linear Optimization.* 1997.

[7] Carlton, W. B. and J. W. Barnes. "Solving the Traveling Salesman Problem with Time Windows Using Tabu Search." *IIE Transactions* 28 (8), pp. 617-630, 1996.

[8] Christof, T. and A. Loebel. Polyhedron Representation Transformation Algorithm (PORTA) software. 1997.

[9] Cline, A. K., D. H. King, and J. M. Meyering. "Routing and Scheduling Coast Guard Buoy Tenders." *Interfaces* 22 (3), pp. 56-72, 1992.

[10] CPLEX Manual, versions 5.0 and 6.0.

[11] Cunningham, W. C., S. T. McCormick, M. Queyranne. "Implementation of a Linear Time Algorithm for Certain Generalized Traveling Salesman Problems." *Integer Programming and Combinatorial Optimization. 5th International IPCO Conference Proceedings.* p. 316-329, 1996.

[12] Dumas, Y., J. Desrosiers, E. Gelinas, and M. M. Solomon. "An Optimal Algorithm for the Traveling Salesman Problem with Time Windows." *Operations Research* 43 (2), pp. 367-375, 1995.

[13] Freye, D. Volkswagen engineer. Private communications.

[14] Gendrau, M., A. Hertz, G. Laporte, and M. Stan. "A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows." *Operations Research* 46 (3), pp. 330-335, 1998.

[15] Gouveia, L. and S. Voss. "A Classification of Formulations for the (Time-Dependent) Traveling Salesman Problem." *European Journal of Operational Research* 83 (1), pp. 69-82, 1995.

[16] Kindervater, G., J. K. Lenstra, and M. Savelsbergh. "Sequential and Parallel Local Search for the Time-Constrained Traveling Salesman Problem." *Discrete Applied Mathematics* 42 (2-3), pp. 211-226, 1993.

[17] Langevin, A. and F. Soumis. "Classification of Travelling Salesman Problem Formulations." *Operations Research Letters* 9 (2), pp. 127-132, 1990.

[18] Langevin, A., M. Desrochers, J. Desrosiers, S. Gelinas, and F. Soumis. "A Two-Commodity Flow Formulation for the Traveling Salesman and the Makespan Problems with Time Windows." *Networks* 23 (7), pp. 631-640, 1993.

[19] Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* New York: Wiley, 1985.

[20] Malandraki, C. and M. S. Daskin. "Time Dependent Vehicle Routing Problems: Formulations, Properties, and Heuristic Algorithms." *Transportation Science* 26 (3), pp. 185-200, 1992.

[21] Mukund, V. *A Decomposition-Based Approach to the Traveling Salesman Problem with Time Windows.* University of Tennessee-Knoxville Ph.D. Thesis, 1995.

[22] Myron, D. L. *Paint Blocking in Ford's In-Line Vehicle Sequencing Environment.* MIT Master's Thesis, Sloan School of Management and Department of Electrical Engineering and Computer Science, 1996.

[23] Nemhauser, G. and L. A. Wolsey. *Integer and Combinatorial Optimization.* New York: Wiley, 1988.

[24] Nygard, K. E. and C. Yang. "Genetic Algorithms for the Traveling Salesman Problem with Time Windows." *Computer Science and Operations Research: New Developments in their Interfaces,* O. Balci, R. Sharda, and S. A. Zenios, eds. pp. 411-424, 1992.

[25] Papadimitriou, C. H. and M. Yannakakis. "The Traveling Salesman Problem with Distances One and Two." *Mathematics of Operations Research* 18 (1), pp. 1-11, 1993.

[26] Pesant, G., M. Gendreau, J. Potvin, J. Rousseau. "An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows." *Transportation Science* 32 (1), pp. 12-29, 1998.

[27] Savelsbergh, Martin. "Local Search in Routing Problems With Time Windows." *Annals of Operations Research* 4, 1985.

[28] Solomon, M. M. and J. Desrosiers. "Time Window Constrained Routing and Scheduling Problems." *Transportation Science* 22 (1), pp. 1-13, 1988.

[29] Tsitsiklis, J. N. "Special Cases of Traveling Salesman and Repairman Problems with Time Windows." *Networks* 22 (3), pp. 263-282, 1992.

[30] Vander Wiel, R. J. *A Decomposition Approach for the Time-Dependent Traveling Salesman Problem.* University of Illinois at Urbana-Champaign Masters Thesis, 1993.

[31] Van Eijl, C. *A Polyhedral Approach to the Discrete Lot-Sizing and Scheduling Problem.* Eindhoven University of Technology Ph.D. Thesis, 1996.

[32] Vander Wiel, R. J., and N. V. Sahinidis. "Heuristic Bounds and Test Problem Generation for the Time-Dependent Traveling Salesman Problem." *Transportation Science* 29 (2), pp. 167-183, 1995.

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ COPIES: (Archives) Aero (Dewey) Eng Hum
Lindgren Music Rotch Science

TITLE VARIES: ▶ ☐ _____

NAME VARIES: ▶ ☐ _____

IMPRINT: (COPYRIGHT) _____

▶ COLLATION: 114 p

▶ ADD: DEGREE: _____ ▶ DEPT.: _____

SUPERVISORS: _____

NOTES:

cat'r: date:

page: J216

▶ DEPT: O.R.

▶ YEAR: 1999 ▶ DEGREE: Ph. D.

▶ NAME: SOKOL, Joel Scott