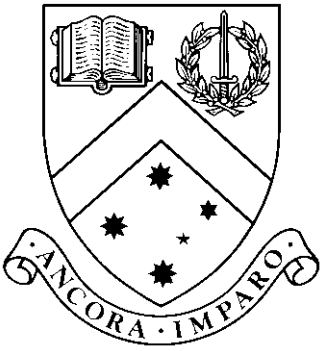
Hybrids of Stochastic Metaheuristics and Constraint Programming for Combinatorial Optimization

by

Dhananjay Thiruvady, BCompSc (Hons.), MIT (Research)



Thesis

Submitted by Dhananjay Thiruvady

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisors: A/Prof. Bernd Meyer, Dr. Andreas Ernst

Clayton School of Information Technology
Monash University

January, 2012

Copyright notices

Hybrids of Stochastic Metaheuristics and Constraint Programming for Combinatorial Optimization

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Dhananjay Thiruvady
January 13, 2012

To my parents - Meera and Vijay Thiruvady

Contents

List of Tables

List of Figures

Vita

During the course of the thesis, the following papers were published. The first two papers were obtained from the results of the chapter on case studies involving car sequencing and single machine job scheduling. A single journal article combining these results with the result from resource constrained mutiple machine job scheduling as a new application is currently being written.

Thiruvady D, Meyer B, Ernst A, 2011 Car Sequencing with Constraint-Based ACO, In *Genetic and Evolutionary Computational Conference*. ACM, Dublin, Ireland, pp. 163–170.

Thiruvady D, Blum C, Ernst A, Meyer B, 2009 Hybridizing Beam-ACO with Constraint Programming for Single Machine Job Scheduling. *Lecture Notes in Computer Science*. 5818:30–44.

Lopez-Ibanez M, Blum C, Thiruvady D, Ernst A, Meyer B, 2009 Beam-ACO based on Stochastic Sampling for Makespan Optimization Concerning the TSP with Time Windows, *Lecture Notes in Computer Science*. 5482:97–108.

Thiruvady D, Meyer B, Ernst A, 2008 Strip Packing with Hybrid ACO: Placement Order is Learnable. In *IEEE Congress on Evolutionary Computation*, IEEE, Hong Kong, pp. 1207–1213.

Thiruvady D, Georgiou-Karistianis N, Egan G, *et al.* 2007 Functional connectivity of the prefrontal cortex in Huntingtons disease. Journal of Neurology, Neurosurgery, and Psychiatry. 78:127–133.

Permanent Address: Clayton School of Information Technology
Monash University
Australia

This thesis was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Glenn Maughan and modified by Dean Thompson and David Squire of Monash University.

Acknowledgments

First and foremost, I would like to thank my supervisors A/Prof Bernd Meyer and Dr. Andreas Ernst. It has been a pleasure to learn from and work with creative individuals such as themselves. I have learnt a great deal from the thoroughness with which they carry out their work. I am also grateful to Prof. Kim Marriott who supervised me during the absence of my supervisors and Dr. Christian Blum who was my supervisor at Universitat Politècnica de Catalunya while I visited Barcelona.

I was fortunate to be able to discuss my work with various academics and post-graduates at Monash. They have always given me constructive feedback related to various parts of my work. Dr. C. Mears, O Woodberry, C. Goncu and Dr. R. Rafeh were particularly helpful. I am also grateful to a number of others, not listed here, who have provided valuable feedback involving different parts of this thesis.

Working at CSIRO Mathematics, Informatics and Statistics gave me the opportunity to interact with experts on various aspects of optimization. It was great to be able to informally discuss any research related topic with them. In particular, Dr. G. Singh and Dr. G. Robinson have provided useful guidance with various topics related to optimization and statistics.

I would like to acknowledge scholarships given by Clayton School of Information Technology, Monash University and the top up scholarship given by CSIRO Mathematics, Informatics and Statistics to support this work. I have also been fortunate to receive various grants to attend conferences overseas aided by my supervisors.

I am grateful to the CSIT and CMIS administrative and technical staff who patiently helped me with administrative issues. I would also like to thank the various office-mates I had over the last five years who maintained an entertaining environment (especially Marc Cheong) and who helped motivate myself during challenging times.

Last but certainly not least, I would like to thank my parents, my brother and Amiza Amir for their love and support.

Dhananjay Thiruvady

Monash University
January 2012

Hybrids of Stochastic Metaheuristics and Constraint Programming for Combinatorial Optimization

Dhananjay Thiruvady, BCompSc (Hons.), MIT (Research)
Dhananjay.Thiruvady@infotech.monash.edu.au
Monash University, 2012

Supervisor: A/Prof. Bernd Meyer, Dr. Andreas Ernst
Bernd.Meyer@infotech.monash.edu.au, Andreas.Ernst@csiro.au

Abstract

Combinatorial optimization problems are frequently encountered in scientific and industrial applications. A large number of these problems are known to be NP-hard and solving these problems by exact/complete methods is often impractical. Therefore, heuristic methods are often the best way to deal with such problems. However, when these problems additionally include hard constraints, heuristic/metaheuristic methods often fail and perform poorly. A solution to this is to integrate these methods with techniques like constraint programming (CP) where constraints may be easily specified and efficiently dealt with thereby providing effective solutions to combinatorial optimization problems with non-trivial hard constraints.

This thesis specifically investigates the integration of stochastic metaheuristics and constraint programming for combinatorial optimization problems (COPs) with non-trivial hard constraints. Among stochastic metaheuristics, we consider ant colony optimization (ACO) and beam search. The base algorithm we begin with is the hybrid CP-ACO (Meyer and Ernst, 2004) and the aim of the thesis is to show that this algorithm can be made efficient by parallelizing the solution construction of ACO (dependent solution construction as opposed to solution construction on multiple machines/cores) via beam search leading to the hybrid CP-Beam-ACO.

We consider three case studies to demonstrate the effectiveness of the new hybrid. The first problem is single machine job scheduling with sequence dependent setup times (SMJS). The aim here is to minimize makespan making sure hard release and due time constraints are not violated. Secondly, a resource constrained multiple machine job scheduled (MMJS) problem is tackled. Here, constraints include release times, due times, precedences, and a resource constraint across the machines. The aim is to minimize the total weighted tardiness of the scheduled jobs. The last

problem considered is the car sequencing (CS) problem. Here a number of cars requiring options must be sequenced such that sub-sequences of a particular length may only allow a specific number of each option. We investigate the performance of our algorithms on the optimization version which further requires the utilization of options be effectively modulated across the sub-sequences.

By applying ACO, Beam-ACO, CP-ACO and CP-Beam-ACO to the above problems we see that CP-ACO methods are by far the best performing algorithms in terms of solution quality. This is clearly seen on the SMJS and CS problems for which complex CP models have been defined. There is a slight disadvantage to CP-ACO for the MMJS problem where the CP model is relatively simple. In terms of finding feasible solutions, the feasibility advantages of CP-ACO are inherited by CP-Beam-ACO. We also see for the MMJS problem that CP-Beam-ACO has a significant advantage in this respect.

In conclusion, this thesis demonstrates that constraint-based ACO is an efficient and effective method to tackle real-world COPs. In particular, CP-ACO can be made efficient by parallelizing the solution construction of the ACO component resulting in CP-Beam-ACO. The new algorithm provides significant advantages for three different types of real-world COPs with non-trivial hard constraints and can be viewed as the practically viable option for implementing ACO with CP.

Glossary and Notation of Terms

Notation

- π: a vector of variables representing a permutation
- $\hat{\pi}$: waiting list of jobs
- $\sigma(\pi)$: a placement obtained from the permutation π
- $f(\pi)$: objective value of the permutation π
- τ: pheromone trails
- η: static heuristic factor
- α: bias for the pheromone, i.e, pheromone raised to α, τ^{α}
- β: bias for static heuristics, i.e, heuristic raised to β, η^{β}
- θ: beam width
- μ: multiplier for beam width
- ϕ: estimate for beam component
- cf: convergence factor
- PR: the precedence set for the jobs in $\mathcal{J}$

Algorithms

- **ACO**: ant colony optimzation
  - **ACS**: ant colony system, a variant of ant colony optimization
  - $\mathcal{MM}$**AS**: $\mathcal{MAX}$–$\mathcal{MIN}$ ant system, a variant of ant colony optimization
- **CP**: constraint programming, refers to FDCP in this thesis
- **FDCP**: finite domain constraint programming
- **CP-ACO**: hybrid ant colony optimization with constraint programming

  - **CP-ACS**: hybrid ant colony system with constraint programming
  - **CP-$\mathcal{MM}$AS**: hybrid $\mathcal{MAX}$–$\mathcal{MIN}$ ant system with constraint programming
- **Beam-ACO**: hybrid beam search and ant colony optimization
  - **Beam-ACS**: hybrid beam search and ant colony system
  - **Beam-$\mathcal{MM}$AS**: hybrid beam search and $\mathcal{MAX}$–$\mathcal{MIN}$ ant system
- **CP-Beam-ACO**: hybrid beam search and ant colony optimization with constraint programming
  - **CP-Beam-ACS**: hybrid beam search and ant colony system with constraint programming
  - **CP-Beam-$\mathcal{MM}$AS**: hybrid beam search and $\mathcal{MAX}$–$\mathcal{MIN}$ ant system with constraint programming
- **LP**: linear programming
- **ILP**: integer linear programming
- **BRIC**: blocked recursive image computation
- **SA**: simulated annealing

Problem Specific Notation

- **SPP**: strip packing problem
- **BP**: bin packing
- **SMJS**: single machine job scheduling with sequence dependent setup times
- **MMJS**: resource constrained multiple machine job scheduling
- **CS**: car sequencing
  - *uoa*: upper over-assignment
  - *uua*: upper under-assignment
  - **DSU**: dynamic sum of utilisation
- **TSP**: the travelling salesman problem

Chapter 1

Introduction

Combinatorial optimization problems (COPs) are important in scientific and industrial applications. These problems commonly appear in practical settings such as timetabling, airline/train scheduling, rostering, network design, computational biology, etc. and can be notoriously hard to solve. Thus, methods that can deal with such problems are of significant interest to scientific and industrial communities which has provided strong motivation to improve existing methods or develop new methods.

Typically, there may be several solutions that solve a COP. The aim is to identify the solution (which may not be unique) that optimizes some measure. Often, the solution spaces associated with these problems are very large (beyond a polynomial factor of the size of the input) with no directed way of exploring them. Thus, it is infeasible to exhaustively explore all solutions.[1] Such hard problems belong to the class of $\mathcal{NP}$-hard or $\mathcal{NP}$-complete problems (Papadimitriou and Steiglitz, 1998; Lewis and Papadimitrious, 1981; Sipser, 2005) and can be distinguished as follows. If the problem in consideration can be reduced to a problem known to be in $\mathcal{NP}$ in polynomial time, the problem is $\mathcal{NP}$-complete. If this mapping is not possible and the problem is still considered as hard as any problem in $\mathcal{NP}$ then it belongs to the class of $\mathcal{NP}$-hard problems. These problems usually require significant resources to identify an optimal solution.

Methods to solve COPs can be broadly categorised as complete/exact or incomplete methods. Complete methods are those methods that explore the entire search space to identify the optimal solution to the problem. However, for large search spaces these methods are often impractical due to time and resource requirements. Exact methods such as constraint programming (CP) (Marriott and Stuckey, 1998) eliminate parts of the search space where no solutions exist so that good solutions

[1] Problems for which a solution can be determined in polynomial time are considered to be in the class $\mathcal{P}$ whereas those problems for which solutions may be validated in polynomial time belong to the class $\mathcal{NP}$. Furthermore, $\mathcal{P} \subset \mathcal{NP}$.

may be obtained more quickly thus making the search more efficient. However, the performance of these methods are dependent on the problem characteristics. For example, tightly constrained problems may be quickly solved with CP whereas the run-time on loosely constrained problems me be prohibitive. In general, full searches with these methods are usually expensive but they will guarantee finding the global optimum.

Solving $\mathcal{NP}$-hard problems to optimality in polynomial time is not possible if $\mathcal{P} \neq \mathcal{NP}$. Hence, complete or exact methods may be infeasible if search spaces are large and there may not be effective ways to systematically reduce them. Thus, alternative methods with reasonable solution times for these problems have been investigated. For example, heuristic methods provide a means to solve these problems in reasonable time-frames using limited resources (Blum and Roli, 2003). These methods do not guarantee finding the optimum but, if designed well, can usually find good locally optimal solutions. The methods are also mostly problem specific, therefore, applying such techniques to new problems are limited.

Metaheuristics (Blum and Roli, 2003) are an alternative that encompass the advantages of heuristics and aim to be problem independent. They achieve this by providing a generic framework which may be adapted to the particular problem being solved. All metaheuristics are designed to intelligently explore search spaces by exploring 'good' regions and provide the best solution until some termination criterion is met (e.g. time limit). This usually results in finding good solutions, with longer searches likely to find better solutions.[2] Among metaheuristics, a popular nature-inspired method is ant colony optimization (ACO) which is the stochastic algorithm focused on in this thesis.

Real-world COPs often consist of non-trivial hard constraints. Metaheuristics have previously shown to be able to deal with trivial constraints, however, their performance on problems with non-trivial constraints is not as effective. In this context, complete/exact and incomplete integrations have led to successful hybrids in the past. Examples include genetic algorithms with linear programming, mixed integer programming, and branch & bound (Puchinger and Raidl, 2005; Klau, Ljubi, c Moser, Mutzel, Neuner, Pferschy and Weiskircher, 2004; Vasquez and Hao, 2001; Talukdar, Baeretzen, Gove and de Souza, 1998).

CP is a technique designed to deal with constraints. However, a potential drawback of CP is when a problem consists of a large search solution space with relatively few constraints. Here, despite the constraints, the search space is so large that finding the optimal solution may be intractable. The search component is where

[2]Most incomplete methods are also called any time methods since they can provide solutions at any time during the search.

metaheuristics are effective. Thus, by combining CP and metaheuristics, their relative advantages can make the resulting hybrid effective on problems with non-trivial constraints and large search spaces and this has indeed been demonstrated previously (Meyer and Ernst, 2004). This thesis examines this CP-ACO hybrid and aims to improve its performance across problem types by addressing some of its inherent inefficiencies.

In ACO, a solution to a problem is constructed from scratch by incrementally adding solution components to partial solution until a complete solution is obtained. Thus, due to its constructive nature, it provides a promising framework to hybridize with CP. Meyer & Ernst (2004) showed how this can be done effectively by applying the ensuing algorithm to a single machine job scheduling problem with sequence dependent setup times. However, their method suffered from the problem of large CPU time requirements in order to solve relatively large problem instances. The main problem here being that the ACO component of the algorithm frequently reconstructs (parts of) a solution requiring repetition of the same propagation steps. Here, we propose a solution to this by further hybridizing with beam search (Blum, 2005) to overcome this problem. Beam-ACO employs a parallel and dependent construction of solutions at each iteration in the style of beam search. Through the partial parallelization of the search it allows the storage of intermediate solver states compactly in a systematic form. This efficient use of the constraint solver leads to more computation time that can be devoted to the optimization/learning component. The resulting algorithm CP-Beam-ACO is shown to be effective on three different problems including single and multiple machine job scheduling and car sequencing.

1.1 Combinatorial Optimization

Combinatorial optimization refers to the theory that deals with methods to solve combinatorial problems (Papadimitriou and Steiglitz, 1998; Christofides, Mingozzi, Toth and Sandi, 1979; Foulds, 1984). These problems are referred to as combinatorial since several aspects of the problem involve discrete structures or components. A COP is an extension of a combinatorial problem where the aim is to optimize an objective. A large number of real-world problems or their abstracted versions can be formulated as combinatorial problems or COPs. Hence, methods for solving these problems have received a lot of attention and a wide range of techniques to tackle such problems exist today. Examples of these techniques include integer programming, graph algorithms, dynamic programming, branch & bound, etc.

A solution to a COP can be represented by a set of variables instantiated with values from their domains. These values may be finite or countably infinite. A

general version of a COP may be defined as one to minimize

$$f(\pi) \tag{1.1}$$

where π is a vector of variables and

$$f : S \to \mathbb{R} \tag{1.2}$$

The above problem may also include a number of constraints:

$$C = \bigwedge_{i=1}^{n} c_i \tag{1.3}$$

where the constraint set C consist of a number of relations c. We require π_k to be integer valued $\forall k$ in the above definition.

A large number of optimization problems may be framed as *linear programming* problems (Bertsimas and Tsitsiklis, 1997). While linear programming is an example of continuous optimization it plays an important role in COPs (Papadimitriou and Steiglitz, 1998). These programs require that the objective and the constraints be linear but do not impose any discrete restrictions. The linear constraints define a convex polytope with optimal solution/s lying at a vertex of the polytope. Two well known algorithms for solving such problems include the *simplex* (Bertsimas and Tsitsiklis, 1997; Papadimitriou and Steiglitz, 1998) and *interior point* methods (Roos, Terlaky and Vial, 2006). Simplex algorithms examine the vertices of this polytope in some order to find the optimal solution (Papadimitriou and Steiglitz, 1998). This leads to linear programs (LP) having combinatorial structure and in worst case the simplex visits an exponential number of vertices. Alternatively, interior point methods are polynomial time algorithms which typically start within the feasible region and work their way towards to the optimal solution. In practical settings, the simplex algorithm is still an effective method when solving a problem with a small number of variables. However, for a problem with many variables, interior point methods are more effective and is currently an active area of research in optimization research community (Hillier and Lieberman, 2005).

An extension of linear programming is to restrict the variables to discrete values leading to *integer linear* programming or integer programming. Here, we have a COP that is closely related to LP and is generally $\mathcal{NP}$-complete. The corresponding LP for an integer program may be useful when tackling the original problem. For example, consider an integer program which is solved by branch & bound search. Typically, the LP *relaxation* (π_k integer restriction is removed) provides a way to obtain bounds to be able to solve the original integer program efficiently.

Now, we give two examples of well known COPs. Often, such problems may be used to guide the development of solutions when tackling new problems. They may also appear as sub-problems within the problems of interest.

1.1.1 Traveling Salesman

One of the well known $\mathcal{NP}$-complete problems is the traveling salesman problem (Dorigo and Stűtzle, 2004; Michalewicz and Fogel, 2004). This is an important problem since it is closely related to other problems such as scheduling, routing, and transportation. The salesman is required to visit all cities exactly once and return to the starting city completing a tour (see Figure 1.1). There is a cost of travel associated between cities and the tour of minimum length must be found. Formally, given n cities $C = \{c_1, c_2, ..., c_n\}$, edges $E = \{(x, y) : x, y \in C\}$, and distances $D = \{d_{xy} : x, y \in C\}$, minimize the total distance travelled:

$$\sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}} \tag{1.4}$$

where $\forall i : \pi_i, \pi_{i+1} \in E$ and $\pi_n = \pi_1$. If $d_{xy} = d_{yx}$ the problem is called *symmetric*, otherwise, *asymmetric*.

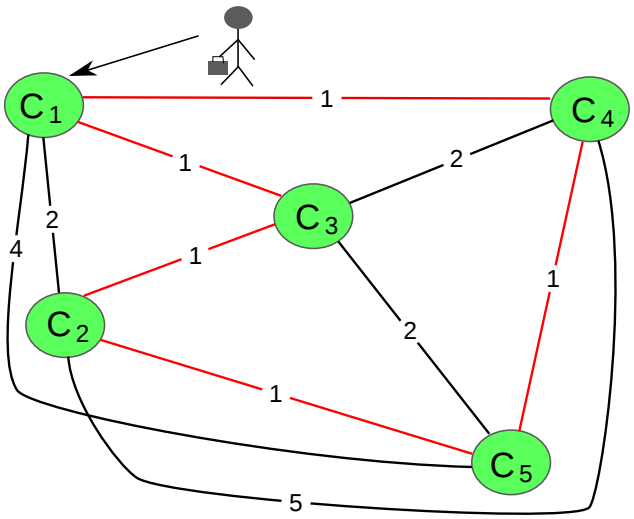


Figure 1.1: The traveling salesman problem; The optimal path is marked in red.

The TSP can be formulated with an integer program as follows (Christofides et al., 1979). Let $x_{ij} = 1$ represent using the edge (i,j) in the tour. The objective is to mimimize

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \times x_{ij} \tag{1.5}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \tag{1.6}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \tag{1.7}$$

$$x_{ij} \in \{0, 1\} \tag{1.8}$$

These constraints ensure that every city is visited only once. We now need to include constraints to ensure a tour is complete, i.e. no sub-tours are generated

$$\sum_{i \in \hat{C}} \sum_{j \in \hat{C}} x_{ij} \geq 1 \qquad \forall \hat{C} \subset C \tag{1.9}$$

Several methods have been applied to solve the TSP and its variants including branch & bound, dynamic programming and branch & cut, etc. See (Applegate, Bixby, Chvátal and Cook, 2007; Reinelt, 2007; Cook, 2011) for an overview of the methods, variants and the state-of-the-art for the problem. The largest TSP instance solved to date is the *World TSP* consisting of 85,900 cities using Concorde (Applegate et al., 2007) with Domino-Parity (Cook, Espinoza and Goycoolea, 2007).

1.1.2 The Assignment Problem

Also called the weighted bipartite matching problem, the assignment problem is another example of a fundamental COP (Papadimitriou and Steiglitz, 1998; Cormen, Leiserson, Rivest and Stein, 2001). Given a bipartite graph, the aim is to assign vertices from one partition to the other in an optimal fashion (see Figure 1.2). The problem occurs in situations like the assignment of task to workers where weights or a cost between the task and worker are given.

The problem may be formulated as a linear program. We are given a complete bipartite graph $G = (U, V, E)$ where tasks $U = \{u_1, \ldots, u_n\}$ and workers $V = \{v_1, \ldots, v_n\}$ represent the two sets of nodes and E represent the edges. This can now be modelled via variables x_{ij}, where if $x_{ij} = 1$ then the edge between v_i and u_j exist in the matching. The objective is to minimize:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \times x_{ij} \tag{1.10}$$

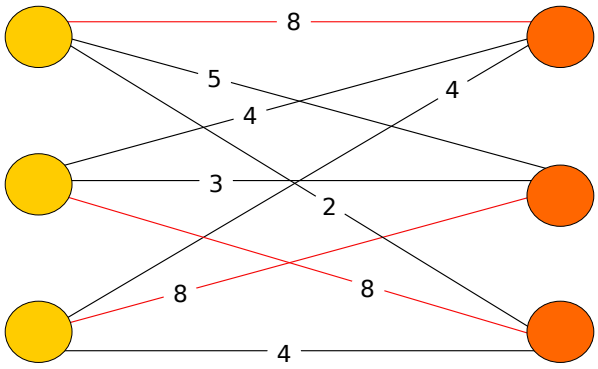


Figure 1.2: The assignment problem with two partitions of three nodes each; The edges in the optimal assignment are marked in red.

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \tag{1.11}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \tag{1.12}$$

$$x_{ij} \in \{0, 1\} \tag{1.13}$$

where w_{ij} is the cost of assigning work i to task j. The above constraints specify that there may only be one element in V that maps to a single element in U. If this formulation is solved with the simplex algorithm for example, the resulting matching will be a real matching without any of the x_{ij}s taking on fractional values. This is because none of the fractional solutions are basic feasible solutions which is the case when the constraint matrix is totally unimodular (Papadimitriou and Steiglitz, 1998). Thus, a solution to this problem may be obtained very efficiently, when at first the problem appears very hard to solve.

Often, some subset of the problem constraints may render the problem $\mathcal{NP}$-hard. An alternative problem without these constraints can be defined which is referred to as a *relaxed* problem or *relaxation*. For example, notice that eliminating the sub-tour constraint, Equation 1.9, from the TSP formulation effectively results in the assignment problem. In general, such relaxations can be extremely useful when solving the original problem. For the relaxation of TSP above, if a number of cities have been visited, a lower bound on the cost of the visiting the remaining cities can be obtained efficiently. This bound can the be used to guide the search to determine the order of the remaining cities.

1.2 Motivation

While metaheuristics are effective on $\mathcal{NP}$-hard problems, they typically struggle when COPs contain non-trivial hard constraints. Traditionally, such constraints in COPs have been dealt with through penalty, repair or multi-phase techniques (Coello, 2002). In the first category, the constraints are handled as soft and penalised when optimizing the objective. In the second category, solutions are constructed and those components where the constraints are violated are repaired. Lastly, with multi-phase techniques, the first phase is to identify feasibility followed by focusing on optimality in the second phase. A common aspect of these methods is that they require problem-specific algorithms which in turn require problem-specific tuning (see for example (Coello, 2002)).

An integration of CP and ACO (CP-ACO) was proposed by (Meyer and Ernst, 2004) as an alternative to the above methods. CP is typically effective at dealing with hard constraints and constraint models for different problem types may be specified effectively without a great deal of effort. CP on its own however, is not suited to solving $\mathcal{NP}$-hard problems since the search component is usually computationally intensive. Therefore, ACO with CP guidance is a promising method for COPs with non-trivial hard constraints and Meyer & Ernst (2004) showed that this was the case for a single machine job scheduling problem.

Despite its advantageous characteristics for tightly constrained COPs, CP-ACO inherently contains inefficiencies. ACO solution constructions are incrementally built from scratch and solutions are independent of each other. The result is that the same solution may be repeatedly built and more so when the algorithm has converged to a region of the search space.[3] While, this is not a significant problem for plain ACO, CP-ACO is very inefficient in this context. This is because CP propagation costs are very expensive and the ACO construction scheme causes repeated propagation which is wasteful. Therefore, a construction scheme which constructs solutions in parallel and avoids this repeated propagation is likely to be more efficient than CP-ACO. One such method is beam search (Pinedo, 2005; Valente and Alves, 2008). Here, solutions are constructed in parallel and no single solution is built twice during the construction phase. Furthermore, a hybrid Beam-ACO algorithm was already proposed by (Blum, 2005) which showed that beam search and ACO can be combined efficiently. This effectively allows us to parallelize CP-ACO's construction procedure, we are able to make significant gains in time by using CP propagation for new solutions always. The result is a significantly faster

[3]This is a direct consequence of ACO's construction scheme which will be detailed in the next chapter.

algorithm, CP-Beam-ACO, which provides significant improvements over CP-ACO where CP-ACO is already effective.

1.3 Aims

The main aim of this thesis was to identify means to improve the performance of CP-ACO given its large computational requirements. In order to do this, we observed the large overhead associated with the repeated computation of parts of the search tree where the costs of CP propagation are high and often wasted. Thus, a solution to this was to partially parallelize the solution construction procedure via beam search leading to CP-Beam-ACO. We show that the new algorithm proves to be more efficient than CP-ACO on three different case studies where CP-ACO is initially shown to be effective. All these problems consist of non-trivial hard constraints but the complexities of the CP model vary from very complex (large propagation costs) to weak models (relatively small propagation costs). In particular, this thesis aims to demonstrate the following hypotheses:

- CP-Beam-ACO is a more effective optimizer than CP-ACO
  - Increasing performance improvements with larger CP propagation requirements
  - Choice of estimates for the beam component are crucial
- CP-Beam-ACO inherits the feasibility advantage of CP-ACO
- The CP-based algorithms are generally more effective than the non-CP algorithms (ACO and Beam-ACO)
  - In particular, for problems where complex CP models can be defined

1.4 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 discusses stochastic metaheuristics including ACO and beam search. The application of these methods to the TSP and a variant of the TSP are discussed. This is followed by discussions of how both these methods may be customized for specific problems to be tackled. The next chapter discusses CP and finite domain CP in particular. Here, the characteristics of the CP solver needed for this thesis are also detailed. The background provided in these chapters leads to the main method suggested in this thesis, namely, CP-Beam-ACO in Chapter 4. Here, hybridizations are discussed in detail including

CP-ACO. This is followed by a discussion on CP-Beam-ACO, its characteristics and its advantages over CP-ACO.

The next chapter, Chapter 5 details how CP-Beam-ACO may be applied to the SMJS, MMJS and CS problems. These studies details the differences between the algorithms for these problems and demonstrate the advantages of CP-Beam-ACO. In Chapter 6 we analyse the estimates for the beam component in detail. Here, problem specific bounds are examined to determine their effectiveness in conjunction with CP-Beam-ACO. All the results are tied in together on the final chapter on conclusions and future work (Chapter 7).

Chapter 2

Stochastic Heuristics and Metaheuristics

Since the early 1990s metaheuristics have been popular methods for solving combinatorial optimization problems. The term metaheuristic refers to a high-level search strategy and any such algorithm is expected to be applicable to a wide range of problems. They are an effective way to approach $\mathcal{NP}$-hard problems (Blum and Dorigo, 2004) as they do not exhaustively explore a search space and are incomplete. Hence, they do not guarantee finding a global optimum but usually find "good" solutions quickly. They trade-off solution quality with run-time and usually produce improved solutions with larger run-times. This is a particularly useful characteristic since alternative models such as those from integer programming might take very long to provide solutions.

These algorithms are characterized by their ability to manage a trade-off between *intensification* and *diversification* (Blum and Roli, 2003).[1] Intensification refers to focusing the search around promising areas of the search space. That is, given that a promising area is found the algorithm should efficiently find a local optimum there. *Local search* algorithms, for example, are effective at intensification. However, considering that search spaces can be quite varying with several promising regions, the algorithm must also explore as many of these regions as possible. This is referred to as diversification and is achieved by algorithmic specific mechanisms designed to avoid getting trapped in local minima. Most metaheuristics can manage this trade-off through a combination of parameters, although, these parameters usually need a lot of tuning before they can be effective on a problem set. Additionally, there are generally no good guidelines for doing this.

Metaheuristics may be classified in several ways (see Table 2.1), and here we discuss three of these classifications suggested by Blum & Roli (2003). Firstly,

[1] These terms are also called *exploration* and *exploitation* (Eiben and Schippers, 1998) but have a more restricted meaning.

metaheuristics can be classified as *constructive* or *non-constructive*. In the former category, solutions are constructed incrementally from scratch during each iteration of the algorithm. Examples include ant colony optimization (ACO), the Bayesian optimization algorithm (BOA) (Pelikan, 2005) and estimation of distribution algorithms (Larrañaga and Lozano, 2002). Each of these algorithms build solutions from scratch by incrementally adding components to a partial solution until a complete solution is obtained.[2] In the latter category, an initial solution is usually constructed in some random fashion and it is modified in some way. Examples of non-constructive algorithms include simulated annealing (SA), genetic algorithms and local search algorithms (Aarts and Lenstra, 2003). For example, genetic algorithms usually work with a population of solutions from which *individuals* (complete solutions) are selected for *crossover* or *mutation*.[3] The resultant solutions may be placed in the population replacing less *fit* individuals (e.g., individuals with a higher cost function in a maximization problem).

Table 2.1: Classification of various metaheuristics.

	Simulated Annealing	Genetic Algorithms	Tabu Search	ACO
Constructive	✗	✗	✗	✓
Population	✗	✓	✗	✓
Nature	✓	✓	✗	✓

Secondly, we can distinguish these algorithms based on whether they are *nature* or *non-nature* inspired. ACO (Dorigo and Stűtzle, 2004) or genetic algorithms (Mitchell, 1996) for example, mimic processes in nature and fall into the former class. Other algorithms such as Tabu search (Glover, 1989; Glover, 1990) or variable neighbourhood search (Mladenovic and Hansen, 1997) come from the second class. This classification can lead to ambiguity, for example, hybrid methods could be classified as both nature and non-nature. Furthermore, components of various algorithms may be attributable to inspiration from nature although they are classified as non-nature.

Metaheuristics may also be classified as *population* versus *single point* based search. Algorithms in the former class usually maintain a population of solutions which guide the search in collaborative manner. Several algorithms such as the genetic algorithm or BOA employ a population to guide the search. In genetic

[2]In ACO, the next solution component is selected by sampling a probability distribution.

[3]Crossover is an operator the takes two individuals (*parents*) and builds a third solution (called *offspring*) by selecting components from each of the parents. Mutation operates on single individual and create a second individual by, either randomly or in some pre-defined manner, modifying its components.

algorithms, new solutions are explicitly created from previous ones which are already in the population whereas in BOA statistics over individuals are used to bias probability distributions which in turn generate new solutions. In contrast, single point methods use a single solution and modify this solution through the duration of the search. These approaches are also called *trajectory methods* (Blum and Roli, 2003) and local search methods like simulated annealing (Kirkpatrick, Gelatt and Vecchi, 1983; Cerny, 1985) is an example of such a search method. In its basic form, simulated annealing can be described as follows. A neighbourhood structure is defined for the problem and from a starting solution (randomly generated), the neighbourhood moves are applied to the solution. If the resultant solution is an improvement, then this move is accepted, however, with some probability moves to non-improving solutions are allowed. This probability reduces over the execution of the algorithm (cooling schedule) and the aim of allowing non-improving moves is to allow the algorithm to jump out of local minima. It has been shown that under certain conditions on the cooling schedule that the algorithm converges to the optimal solution as the number of steps of the algorithm approaches infinity (Aarts, Korst and van Laarhoven, 1997).

A number of metaheuristics include a learning component within the algorithm. These algorithms are iterative and the learning usually takes place between iterations. The idea is that the algorithms try to learn characteristics about or dependencies between the variables such that they biased towards promising regions of the search space. Genetic algorithms and ACO are examples of algorithms which learn these characteristics over-time. For example, ACO attempts to learn implicit probability distributions by constructing solutions and biasing the probabilities towards the best solutions found in an iteration. Such learning is also referred to as *reinforcement learning* (Sutton and Barto, 1998).

This thesis focuses on ACO and Beam search, a stochastic heuristic tree search method. ACO is a reinforcement based stochastic metaheuristic and has been shown to be effective for practical problems. Beam search on the other hand is a well known tree search method (Pinedo, 2005). The procedure usually works by considering a fixed number of partial solutions in parallel and continues their construction by using some sort estimate, which is possibly stochastic. Hence, this algorithm can be viewed as a stochastic tree search method. ACO and Beam search can be combined easily in various ways. These details will be discussed in future sections. Here, we point out the ACO can also be viewed as a tree search method where a new node is selected probabilistically and if this is coupled with the heuristic estimates, the resulting algorithm can be considered as a Beam-ACO hybrid. Such hybrid algorithms have also shown to be very effective in practical settings (Blum, 2005; López-Ibáñez, Blum, Thiruvady, Ernst and Meyer, 2009).

2.1 Ant Colony Optimization

ACO is a reinforcement based stochastic metaheuristic inspired by the foraging behaviour of real ants (Dorigo and Stűtzle, 2004). Ants leave their nest looking for food and the lay pheromone on the paths they use when returning to their nest. Other ants will go out looking for food guided by the pheromone by probabilistically following the paths. The pheromones are modulated either by food or path quality which result in shorter paths or paths to better food sources receiving increased deposits. The pheromone concentration also decreases gradually which is referred to as pheromone *evaporation*. This creates a positive feedback loop resulting in the colony choosing the more desirable paths to food sources (Camazine, Deneubourg, Franks, Sneyd, Theraulaz and Bonabeau, 2001).

ACO is also an example of swarm intelligence (Bonabeau, Dorigo and Theraulaz, 1999). In general this refers to the collective behaviour where individuals in the swarm cooperate through self-organization rather than being directed by some centralized control. Various nature-inspired algorithms based on these principles of swarm behaviour have been used to tackle COPs.

The principles underlying the collective behaviour of the colony can be used to solve COPs. Each member of the colony is required build solutions to the problem independently. They make use of artificial pheromone *trails* which are initially unbiased and therefore the solutions are constructed in a uniformly random fashion. The construction itself progresses sequentially where at each step the partial candidate solution is extended by adding a new solution component[4] until a complete solution is obtained. The next component is chosen probabilistically according to the pheromone information. When all the ants have completed their solutions an iteration of the algorithm is complete. Now the solutions available can be used to bias the pheromones towards the "good" solutions seen so far. The algorithm is repeated until some termination criterion (e.g. pre-defined time limit).

The description given above is high-level and when applying ACO to a problem several details need to be considered. These include a the actual pheromone model, parameters (e.g. number of ants), evaporation schemes and update schemes. While there are some guidelines for the way in which these details can be implemented in the case of well investigated problems,[5] usually, problem specific settings need to be determined.

[4]For example, if we consider the TSP, a solution component is a city. Here, at each step, a city is added to a partial list of cities until a complete tour is constructed.

[5]For example, past experiments can suggest parameter settings or a pheromone model for a similar problem might be useful to attempt on the current problem. See (Dorigo and Gambardella, 1997) for an example of the parameter settings for the TSP.

An example of a problem specification and specifying a pheromone model will make things clearer. Consider a problem where n variables $X = \{x_1, \ldots, x_n\}$ have potential domains $x_i \in D = \{0, \ldots, n\}$. A solution is defined as a permutation $\pi(X)$. Without any other knowledge about the problem, a plausible model for the pheromones could be to define τ_{ij} as the desirability of selecting domain value j for component i. The pheromones can then be coupled with problem specific static (usually the case) or dynamic heuristics. Now the modified pheromone trails are used to select a value for each variable. This will be used as a running example through the following sections, where no other ACO model is defined, as a means to illustrate the various procedures/variants discussed.

Algorithm 1 Basic Ant Colony Optimization Framework.

```
1: INPUT: A problem instance, n_a (number of solutions)
2: initialize π^ib (iteration best), π^bs (global best), 𝒯 (pheromones)
3: while termination conditions not satisfied do
4:    S_iter := ∅
5:    for j = 1 to n_a do
6:       π_j := ConstructSolution(𝒯)
7:       if π_j is a feasible solution then S_iter := S_iter ∪ {π_j} endif
8:    end for
9:    // Set the iteration best to the best solution
10:   π^ib := argmin{f(π)|π ∈ S_iter}
11:   if π^ib is a feasible solution then
12:      Update(π^ib,π^bs)
13:      // update pheromone trails with the current best solution
14:      GlobalPheromoneUpdate(𝒯,π^bs)
15:   end if
16: end while
17: OUTPUT: π^bs
```

The generic ACO framework is shown in Algorithm 1 and works as follows. This procedure takes as input an instance of the problem. Two "ants" which represent the best solution found during an iteration (π^{ib}: iteration best) and the best solution found during the execution of the algorithm (π^{bs}: global best) are first initialized. A pre-defined pheromone model for the problem is also initialized. Here, the main iteration of the algorithm commences (line 3). A population of solutions S_{iter} is maintained at each iteration and all feasible solutions built are appended to this population (lines 5-8). The parameter n_a specifies the number of individual solutions to be built during an iteration.[6] The procedure ConstructSolution($\mathcal{T}$) samples the pheromone trails to generate solution components incrementally until a complete solution is built or the solution is rendered infeasible. A value j is selected for component i according to the following distribution:

[6]Note that not all of these solutions are required to be feasible.

$$\mathbf{p}(\pi_i = j) = \frac{\tau_{ij}^{\alpha} \times \eta_j^{\beta}}{\sum_{d \in D \setminus \{\pi_1,\ldots,\pi_{i-1}\}} \left(\tau_{id}^{\alpha} \times \eta_d^{\beta}\right)} \tag{2.1}$$

Static or dynamic heuristics factors can be specified via η. α and β are parameters that can be used to specify the relative contributions of the pheromones/heuristics. The iteration best is set to the best of the feasible set (line 10) and the global best is updated to the iteration best if the iteration best is an improvement upon it. The pheromones are then biased by the global best solution (referred to as *elitist update*) with the call GlobalPheromoneUpdate($\mathcal{T}$,π^{bs}) with the aim being that the search is biased towards the best solution found so far.[7] For example, given a complete feasible permutation π:

$$\tau_{ij} \leftarrow \tau_{ij} \times (1.0 - \rho) + f(\pi) \times \rho \tag{2.2}$$

The first part of the equation is called evaporation ($\tau_{ij} \times (1.0 - \rho)$).[8] The second part is a *reward* that is proportional (inversely proportional for minimization problems) to the cost - $f(\pi)$ - of the solution π. This completes the high-level description of ACO.

ACO can also be represented as a tree search. For example, see Figure 2.1. The example shows how two ants would sample the pheromones (τ) biased by heuristic factors (η) to make a value selection for the second variable during the construction process. The root of the tree represents an empty solution. The two solutions initially have a state where $x_1 = 2$ and $x_1 = 4$ for the first variable. The two figures show how $x_2 = 4$ and $x_2 = 3$ are selected for the two solutions, respectively. This figure also shows solutions are built independently, however, schemes like ant colony system (see later) apply changes to the selection pheromones thereby implicitly affecting solutions future solutions to be constructed. This representation is particularly useful in the context of this thesis since other tree search methods are used and integrations with ACO are more easily explained.

Given the algorithm description above, we now discuss specifics of two variants ACO. At a high level they perform the same actions/steps but the implementation of the steps are quite different. Hence, specific details are discussed in the next section.

[7] Note that this scheme described here is the one used throughout this thesis, however, other variants of ACO like *Ant System* use the entire population to bias the pheromones proportional to the solution qualities.

[8] Reproduces the effect of evaporation of real pheromones that happens over time.

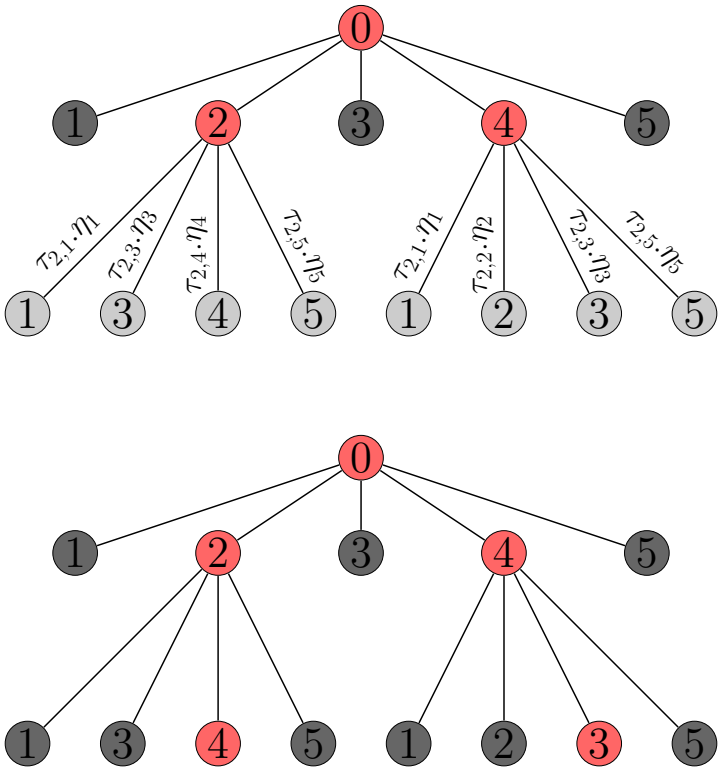


Figure 2.1: This figure demonstrates how ACO can be viewed as a tree search. Node 0 is the root of the tree where the solution construction starts. The first variable is represented at level 1. In this example, two ants are constructing solutions in parallel and have chosen $x_1 = 2, \pi_1 = \{2, \ldots\}$ and $x_1 = 4, \pi_2 = \{4, \ldots\}$, where the index represents a variable. The figure on top shows the pheromones and heuristics factors that are considered by each ant when attempting to select the next component. Given this information, the first ant chooses $x_2 = 4, \pi_1 = \{2, 4, \ldots\}$ and the second ant selects $x_2 = 3, \pi_2 = \{4, 3, \ldots\}$.

2.1.1 ACO Variants

The original variant of ACO was called Ant System (AS) (Dorigo and Stűtzle, 2004). While this algorithm was shown to be effective on certain problems (e.g. subset problems (Leguizamon and Michalewicz, 1999)), including originally the traveling salesman problem (TSP), its performance on larger problems was not competitive (Dorigo and Stűtzle, 2004). Hence, improvements to AS were explored including the *elitist ant system*, *rank-based ant system*, *ant colony system* (ACS), $\mathcal{MAX}$–$\mathcal{MIN}$ *ant system* ($\mathcal{MM}$AS), etc (see (Dorigo and Stűtzle, 2004) for an overview of the variants). Of these variants, ACS and $\mathcal{MM}$AS have been the most popular in practice. Both algorithms use elitist update but vary in the way they are implemented. The alternative to elitist update is to use all solutions to update the pheromones and this scheme was used by the original ant system.

Ant Colony System

ACS has been used in various studies successfully (Dorigo and Gambardella, 1997; Gambardella and Dorigo, 2000; Blum and Roli, 2003). The motivation behind ACS was to improve the poor convergence aspects of AS. In particular, this algorithm makes use of its accumulated search experience more significantly than the AS by emphasising the best solutions found. Algorithm 1 remains unchanged, however, differences appear within components of the procedure. It differs in three aspects from AS:

- Local pheromone update: this is applied to each value that is selected for a solution component. This amounts to:

$$\tau_{ij} \leftarrow \tau_{ij} \times (1.0 - \rho) \tag{2.3}$$

  where ρ is a learning rate parameter which is usually set to a low value. This procedure is applied within the $\mathsf{ConstructSolution}(\mathcal{T})$ procedure in Algorithm 1. The aim behind this procedure is to emphasize diversification such that the ants are biased to pick different values during the solution construction. This bias affects other solutions which are still to bind variable i and hence, the solutions are not built entirely independently.

- Greedy selection: given a parameter q_0, the value for a component is selected deterministically according to the highest pheromone value. q is sampled uniformly from the interval $[0, 1]$:

$$\mathbf{p}(\pi_i = j) = \begin{cases} \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{k \in I_i} \tau_{ik}^{\alpha} \times \eta_{ik}^{\beta}} & if\ q > q_0 \\ max_{k \in I_i}\{\tau_{ik}^{\alpha} \times \eta_{ik}^{\beta}\} & otherwise \end{cases} \tag{2.4}$$

- Elitist update: as specified earlier, ACS uses an elitist update according to Equation 2.2. While this update is usually based on the global best solution (the best solution found up to this point in the search), some applications show improved performance by using the iteration best solution (the best solution constructed during the current iteration, see (Dorigo and Stűtzle, 2004)).

$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System

The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stűtzle and Hoos, 2000) is the other ACO variant used in this thesis.[9] It has also been successfully used in practice

[9] $\mathcal{MM}$AS is chosen for the experiments here as it has been shown to be an effective implementation of ACO and some of the studies we examine have already shown $\mathcal{MM}$AS to be effective.

(Blum, 2005) Like ACS, $\mathcal{MM}$AS build upon the basic principles of AS and the high-level Algorithm 1 is the basic framework used here.

$\mathcal{MM}$AS in its original version differs from the other variants in the following aspects:

- Upper and lower bound on the pheromones: the pheromones have lower and upper limiting values (τ_{min} and τ_{max}) which are specified as parameters to the algorithm. The lower bound is typically set to a value close to 0 but positive to ensure that the algorithm does not get stuck in a local minima. In the hyper-cube framework (see next section) $\tau_{min} = 0.01$ and $\tau_{max} = 0.99$ are usually used to ensure that the values lie between 0 and 1.

- Initial pheromone distribution: the initial values are usually set to be equal to the upper bound (Dorigo and Stűtzle, 2004). In some schemes like (Blum, 2005) other initial conditions are specified.

- Restarts: The algorithm restarts every time the pheromones converge and the ants are almost always constructing the same solutions. The restart schemes are quite varied but depend on a convergence measure[10] which is a function of the pheromones.

- Elitist update: this is identical to what is done with ACS. Updates based on the global best and iteration best have been attempted and there is no consistent way suggested to implement this. In fact, (Blum, 2005) showed that some combination of the iteration best, global best and another solution - the restart best - is the best way to implement this.

At the end of each iteration the pheromone for every component is evaporated similar to Equation 2.3. The pheromone update in Equation 2.2 is accordingly modified to look like:

$$\tau_{ij} \leftarrow min(max(\tau_{ij} \times (1.0 - \rho) + f(\pi) \times \rho, \tau_{min}), \tau_{max}) \tag{2.5}$$

which is essentially the same as the global update with the bounds applied.

In this thesis $\mathcal{MM}$AS is used extensively. However, the details of the specific implementation for each application has varied due to the "ideal" implementation suggested by previous studies. These details are provided in the relevant chapters.

[10]For example, a simple convergence measure that can be implemented can be defined as follows. For a number of iterations, if the algorithm is constructing the same iteration best solution, then the pheromones are likely to have converged and can be reset.

The Hyper-Cube Framework

The hyper-cube framework (HCF) for ACO is applicable to all of the available ACO variants (Blum and Dorigo, 2004). This framework imposes a change to the pheromone update by modifying Equation 2.2:

$$\tau_{ij} \leftarrow \tau_{ij} \times (1.0 - \rho) + \rho \tag{2.6}$$

This results in the pheromone values falling in the interval $[0, 1]$. Practically, HCF has the advantage that this handles the scaling of the objective function. Theoretically, Blum & Dorigo (2004) were able to prove that for the original AS algorithm the expectation of average quality of solutions improves overtime.

2.1.2 Applying ACO

Here, we show an example of how ACO may be applied to a problem. We consider the TSP introduced in Chapter 1 and focus on the study by (Dorigo and Gambardella, 1997) who showed that ACO can be applied effectively to this problem. This paper also considers the ATSP where the distances $d_{ij} = d_{ji}$ does not necessarily hold for any two cities i and j.

For the TSP, a solution can be represented by a sequence or π where the sequence represents the order in which the cities are visited. Here, we require $\pi_1 = \pi_n + 1$ so that the first and the last cities visited to the same city in order to complete the tour. Successor relationships are particularly important for this problem, i.e., we visit a city given the knowledge of the previously visited city. Thus, the pheromone information encodes this bias where τ_{ij} represents the desirability of of visiting city j given that we are at city i. Dorigo and Gambardella (1997) use ACS which makes use of Equation 2.4[11] when selecting any city that is not the starting city. The solution construction procedure initially picks a random city (the original starting location does not matter) and incrementally adds cities until a complete tour is obtained. The next city chosen is based only on the previous city and is selected from the candidate set of cities that have not yet been visited.

The heuristic factors η_{ij} are chosen as $1/d_{ij}$ which is the inverse of the distance between the city to be chosen j to city i that we have previously visited. This biases the selection to favour cities that are close in distance. Intuitively, selecting amongst the closest cities is likely to lead to better solutions and this is encoded by static heuristics.

Once a number of tours have been completed, local search is applied to the best of these solutions possibly improving it. This solution is used for the global update

[11] Note that $\alpha = 1.0$ for this study

of the pheromones using Equation 2.2. Here $f(\pi^{bs})$ is set as:

$$f(\pi^{bs}) = 1/\sum_{i=1}^{n} d(\pi_i^{bs}, \pi_{i+1}^{bs}) \tag{2.7}$$

where π^{bs} is the best sequence found and π_1^{bs} and π_{n+1}^{bs} are the same city. Being ACS, a local update rule is also applied every time a city is chosen. Various methods of implementing this were examined by (Dorigo and Gambardella, 1997) and a modified version of Equation 2.3 is applied

$$\tau_{ij} \leftarrow \tau_{ij} \times (1.0 - \rho) + \tau_0 \times \rho \tag{2.8}$$

where $\tau_0 = 1/(n \times f(\pi_{nn}))$ is the initial pheromone level. π_{nn} is the sequence produced by the nearest neighbour heuristic, i.e., from each city, select the closest city to move to and repeat this until a tour is complete. Note that (Dorigo and Gambardella, 1997) explore other initial values of τ_0 and these details can be obtained from the paper. The learning rate is set to $\rho = 0.1$ and $q_0 = 0.9$ which favours high deterministic selection. Details of the parameter settings choices are also discussed in the paper, but as we mentioned earlier, there is no easy way of choosing them.

Given these settings, Dorigo and Gambardella (1997) show that ACS combined with local search is an effective way to tackle the TSP and the ATSP performing competitively with simulated annealing and genetic algorithms.

2.2 Beam Search

Beam search was first introduced in the 1970's as a alternative to tree search methods (Rich and Knight, 1991). Like other metaheuristics, it is an incomplete heuristic search and provides an effective way to deal with $\mathcal{NP}$-hard problems (Pinedo, 2005; Valente and Alves, 2008). It can be viewed as a branch & bound like search where estimates are used to guide the search through each level of the search tree. At each level of the tree the algorithm keeps track of a maximum number of solutions which are chosen based on the estimates. Since the solutions are limited by this parameter, the algorithm runs in polynomial time given the input. However, unlike branch & bound methods back-tracking is not allowed. Therefore, the algorithm is incapable of retracing its steps to reconsider promising branches of the tree that it might have eliminated incorrectly. However, the basic algorithm can be modifed to explore alternative paths via an iterative procedure that incorporates some variation in the node selection. Beam search has been successfully applied to several COPs and in particular to scheduling (Bautista, Companys and Corominas, 1988; Bautista, Companys and Corominas, 1996; Valente and Alves, 2008).

Algorithm 2 The Basic Beam Search Framework.

1: INPUT: $(\theta, \mu, \mathcal{T})$
2: // initialise the beam with beam width empty solutions
3: $B_0 = \{\pi_1 = (), \ldots, \pi_\theta = ()\}$
4: $t = 0$
5: **while** $t < n$ **and** $|B_t| > 0$ **do**
6: **for** $i \in B_t$ **do**
7: $k \leftarrow 0$, $D = \text{domain}(\pi^i_{t+1})$
8: **while** $k < \mu$ **do**
9: $\overline{\pi} = \pi^i$
10: $j = \text{select\_component}(D, \mathcal{T})$
11: $\overline{\pi}_t = j$
12: $B_{t+1} \leftarrow B_{t+1} \cup \overline{\pi}$
13: $k \leftarrow k + 1$, $D = D \setminus j$
14: **end while**
15: **end for**
16: $B_{t+1} = \mathsf{Reduce}(B_{t+1}, \theta)$
17: $t \leftarrow t + 1$
18: **end while**
19: OUTPUT: $\text{argmax}\{f(\pi) \mid \pi \in B_{n-1}\}$

The search proceeds in a parallel fashion like ACO. However, unlike ACO, there are dependencies between the solutions. The general framework for Beam search is as follows (see Algorithm 2). A data structure called the *beam* (B) consists of a set of partially completed solutions to be constructed. The algorithm builds the solution by binding values to variables incrementally (lines 5-17). Initially, the beam consists of no solutions. μ potential extensions are generated to fill the beam (lines 8-14) for each partial solution π^i_{t+1}. This can be done in several ways, for example, with a greedy selection based on some static heuristic. From amongst these candidates, a pre-defined number - the beam width θ - children are selected to be placed in the beam (line 16). If there are fewer candidates than this number to select from all of them are selected.

Consider Figure 2.2. This example shows a single level of construction in the Beam search procedure. Two solutions are being built in parallel: $\pi^1_2 = \{2, \ldots\}$ and $\pi^2_2 = \{4, \ldots\}$. To select a value for the next variable, a fixed number of extensions for each solution are considered. Candidate extensions 3, 4 and 5 for the first solution and 1,2 and 3 for the second solution are considered.[12] An estimate (ϕ) for each child is computed (e.g. $\phi_{2,3}$ for candidate 3 given that 2 was the previously bound variable) and two extensions for each partial solution are selected greedily based on these estimates. Here, we can see how the dependence between solutions in the beam cause the branch of the tree $\pi^1_2 = \{2, \ldots\}$ to be eliminated from the search.

[12]Candidate 2 is ruled out for the first soltuion since it has alread been bound to variable 1.

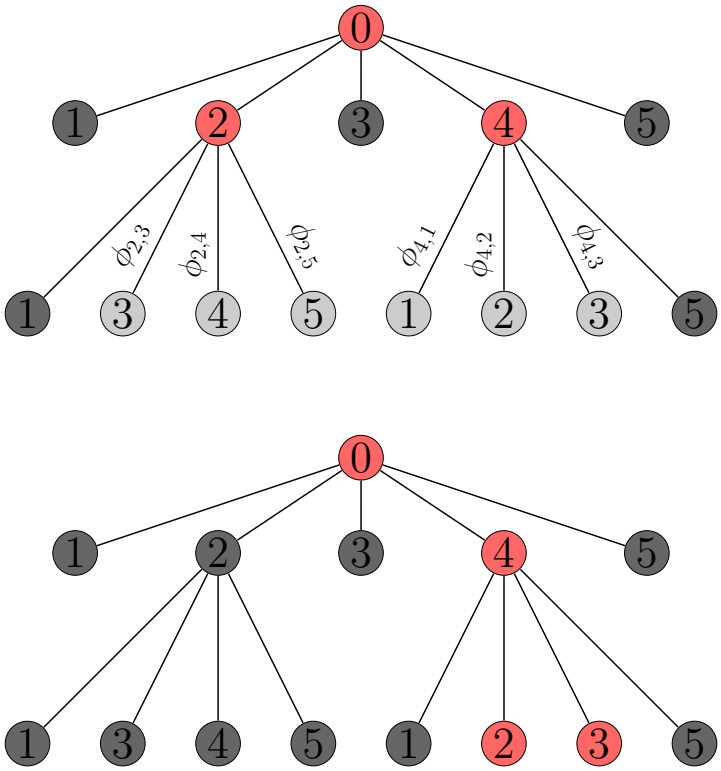


Figure 2.2: The Beam search procedure is shown here for two variables. Node 0 is the root of the tree where the solution construction starts. The first variable is represented at level 1. The following parameters are set: $\theta = 2$, $\mu = 3$. Two solutions are constructed in parallel and $x_1 = 2, \pi_2^1 = \{2, \ldots\}$ and $x_1 = 4, \pi_2^2 = \{4, \ldots\}$ have been chosen for the first variable. The figure on top shows that for each child, 3 solution extensions are considered for which estimates (ϕ) are computed. Using the estimates, the solutions chosen to be placed in the beam are $\pi_3^1 = \{4, 2, \ldots\}$ and $\pi_3^2 = \{4, 3, \ldots\}$.

The search instead focuses on the branch with $\pi_2^2 = \{4, \ldots\}$. Therefore, it is vital that ϕ is chosen appropriately, otherwise, the search will be guided into sub-optimal regions of the search space. This also provides the added avantage of focusing the search more stongly around optimal regions if the estimates are effective.

The estimate can be obtained in several ways but an effective estimate is crucial to the success of the search. We can distinguish between estimates that compute information about a child at the next level of the tree (*priority evaluation*) and those that compute information about a child extrapolated through to complete solution (*total cost evaluation*) (Valente and Alves, 2008). While priority evaluation is usually very quick its local nature may be misleading at the global level. Since the estimate is directly used to compare solution extensions, an estimate based on a single variable is unlikely to provide enough information about how good a complete solution is. Therefore, the resulting estimate may actually prune away more promising nodes of the search tree.

Hence, the preferred option is usually total cost evaluation methods which may incur a significantly larger cost but can be designed to guide the search with the global goal in mind. A common method to obtain the estimate is by using known bounds for a problem. Typically, a relaxed version of the problem may permit a quick bound which can be substituted for the estimate. See for example Valante *et al.* (2008) who explore a Beam search algorithm for a single machine total weighted tardiness problem with sequence dependent setup times. Problem specific estimates are discussed later in the thesis when needed, but in the next section we discuss *stochastic sampling* or *probing* which is consistently examined for all problems to follow.

In its classical version, Beam search does a single iteration through all the variables in the problem terminating the search when it reaches the leaves of the tree. However, often the estimates might lead to sub-optimal regions and this leads to the idea of repeating the Beam search procedure with a bias towards the better solutions observed thus far. This leads to an obvious integration which is discussed next.

2.2.1 A Beam-ACO Hybrid

The Beam search procedure can benefit from repeated iterations of the search with a learning component. By introducing some variation in the candidate selection the search can be guided out of local optima. Such a learning component is the basis of ACO which makes a Beam-ACO integration an obvious choice. In fact, it was originally shown by Blum (2005) that this integration is very effective on open shop scheduling problems and other studies have substantiated this (López-Ibáñez et al., 2009). The basic idea is to use ACO as a high-level framework (can be implemented with any of the variants) and to replace the solution construction (line 6 in Algorithm 1) by a probabilistic version of Beam search. Thus, a probabilitic model for learning such as the pheromone model in ACO is suitable for this purpose. Once this is defined, the children can be selected using one of the ACO selection schemes which will be implemented in Algorithm 2. The candidate soltuions can now be reduced to θ solutions based on the estimate (line 16 in Algorithm 2). This probabilistic variant initially conducts an unbiased search when selecting children but then biases the search towards promising areas through the learning component. Coupled with the estimates, Beam-ACO incorporates two levels of selection providing mroe flexibility than plain ACO.

This implementation of Beam search also permits an alternative way to determine a total cost evaluation estimate. The basic idea is to use the ACO solution construction scheme which is relatively cheap. This is discussed next.

Stochastic Sampling

Probing was first introduced in the context of heuristic tree search methods for boolean satisfiability and number partitioning (Ruml, 2001) . Here, we consider stochastic sampling, which can be considered as a way to achieve probing. The idea behind this method is relatively simple. Instead of deriving an independent estimate for Beam search, an estimate may be obtained by completing a number of solutions (N^{s}) randomly and choosing the cost of the best of these solution to be the estimate. In the context of Beam-ACO, the basic ACO construction procedure may be used to generate the N^{s} samples from which the cost of the best of these is used as the estimate. Alternatively, if an even quicker estimate is desirable, a solution may be completed deterministically using the greedy component of Equation 2.4 for every variable until the solution is complete. If heuristics are used, then in the initial part of the search this estimate basically evaluates to the static heuristic and at later stages of the search it is biased entirely by the pheromones. Although these methods of obtaining estimates are relatively simple they has shown to work effectively for some scheduling problems (López-Ibáñez et al., 2009; Thiruvady, Blum, Meyer and Ernst, 2009).

2.2.2 Applying Beam-ACO

Beam-ACO may be applied to problems where ACO can be applied effectively. However, its performance is dependent on the quality of the estimates that are obtained for the beam search component of the algorithm. Coupled with the learning component, Beam-ACO is more flexible than plain ACO with intensification or diversification more easily achieved. However, the drawback is that additional parameters have been introduced leaving the algorithm designer with with further choices to make which may not be easily achieved.

Here we show that Beam-ACO can be applied effectively to a variant of the TSP problem. We consider the TSP with time windows (TSPTW) which is effectively the TSP that we discussed earlier with the additional constraints that each city i has an associated start time s_i and end time e_i specifying the time window when the city must be visited. The constraints are implemented as hard, where if we have arrived early we can wait but are not allowed to arrive late.

The problem above was tackled by (López-Ibáñez et al., 2009) where we used Beam-ACO similar to Algorithm 2. ACS was the variant used for the learning component. A solution can be represented by the sequence π and the beam is composed on θ of these partial sequences at any one time. For every solution in the beam, we consider all possible children and select a subset of these ($\theta \times \mu$) in a

similar fashion to Equation 2.4 where the heuristic η_{ij} is based on the earliest start time (s_j) and latest end times (e_j):

$$\eta_{ij} = \frac{1.0}{\lambda^d \times \frac{d^{\max} - d_{ij}}{d^{\max} - d^{\min}} + \lambda^e \times \frac{e^{\max} - e_j}{e^{\max} - e^{\min}} + \lambda^s \times \frac{s^{\max} - s_j}{s^{\max} - s^{\min}}} \tag{2.9}$$

and $\lambda^d + \lambda^e + \lambda^s = 1$ are weights that balance the impact the heuristics. The above measure essentially provides information combining the travel costs between cities. See (López-Ibáñez et al., 2009) for further details.

Stochastic sampling is used to obtain the estimates. For each partial sequence (amongst the $\theta \times \mu$ candidates), a number of samples are completed using Equation 2.4 with $\alpha = 1.0$. The cost of the best of these samples is used as the estimate. In line 14 of Algorithm 2, we reduce the set of partial solutions to consist of θ solutions and we make this choice greedily based on the estimate. Stochastic sampling provides two estimates here (1) violations of the time windows and (2) estimate of solution quality. θ solutions are chosen greedily by first considering feasibility and then solution quality.

We see from this study that Beam-ACO based on stochastic sampling can be applied effectively to problems where ACO may be applied. Furthermore, the results obtained here are much improved compared to plain ACO and other metaheuristics. A critical aspect of the algorithm is its ability to deal with hard constraints and therefore more effective on tightly constrained COPs than ACO. Beam-ACO is aided by the estimates obtained from stochastic sampling.

2.3 Customizing ACO and Beam-ACO

Often, customizing ACO or Beam-ACO for a problem can be problematic. In fact, metaheuristics in general, can be difficult to customize but once this is done they can be very effective. The problems based on the TSP discussed earlier have been given a lot of attention and the results clearly show that metaheuristics are useful for these problems. Here, we take a step further and show that metaheuristics can be effective for other types of problems. Through these studies we confirm metaheuristics' ability to learn orders and that they can also effectively incorporate problem-specific heuristics in their learning schemes.

We consider the example of strip packing (Martello, Monaci and Vigo, 2003) where (Hopper and Turton, 2001) suggested that order learning by metaheuristics may be ineffective for this problem. By carefully customizing ACO for this problem, we show that this is not the case and that order learning can in fact lead to improved packings. Details of the complete study can be found in (Thiruvady, Meyer and

Ernst, 2008). We briefly describe the problem, outline the results and discuss main conclusions here.

2.3.1 Customizing ACO

The strip packing problem (SPP) requires the placement of a number of items on a strip of fixed width but infinite height (Martello et al., 2003). Here rectangles of fixed width and height must be placed on the strip without overlapping and the aim is to minimize the height of the strip. The items can be rotated by 90° to potentially achieve a more dense packing. Guillotineable packings (Dychhoff, 1990) are not required. This problem was shown to be $\mathcal{NP}$-complete (Fowler, Paterson and Tanimoto, 1981) but like other similar problems in operations research, a simple heuristic is effective in solving it (Hopper and Turton, 2001).

To place items, (Hopper and Turton, 2001) showed that bottom-left fill heuristic (BLF) is the most effective method. The BLF decoder works as follows. Starting at the bottom-left of the strip the decoder moves right checking potential locations that an item could fill. If no suitable location is found the decoder increments the height by one, starts from the left and moves right attempting to place the item. Figure 2.3 shows the candidate location for an item to be placed. Now two items are placed. The first item (item 5) is placed in the lowest possible location on the strip while the second item (item 6) is able to fill the gap with this scheme.

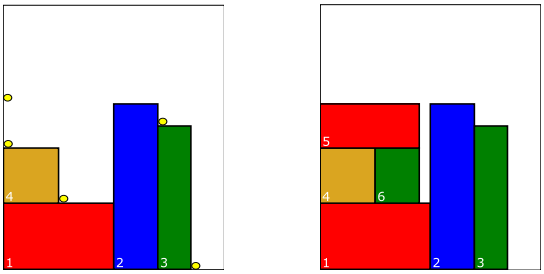


Figure 2.3: The BLF placement heuristic; The strip on the left shows a partial layout with BLF placement points, the small circles show candidate locations for the next item; Two items are placed according to the BLF placement scheme. This example demonstrates how BLF is able to effectively fill the gap that was created by item 5.

The variant chosen for this study was ACS. The procedure at a high-level is to construct a number of permutations using the pheromone trails and then pack the items according to their orders using the BLF heuristic. The permutation for the best placement is used to update the pheromones and the procedure repeats for a number of iterations, see (Thiruvady et al., 2008) for these details.

With respect to learning there are two separate aspects that can be considered: (1) orders and (2) rotations. Thus we conducted experiments consisting of the learning of item order, item order and rotations independently, item order and rotations dependent on an item's position in the order and item order and attempting both rotations.

Results and Discussion

The full set of results are available in Appendix B.1 and are summarized here. Learning orders are always advantageous over no learning. This is true in the cases where both orientations are attempted and also with learning orientations. Surprisingly, learning orientation on its own does not seem to benefit the placement generally. For the smallest instances it seems to make a difference but it fails to make a difference with larger instances. In Appendix B.2 we see an example of a packing on a large problem obtained with and without learning. Clearly, learning has a significant role.

Comparing with other metaheuristics implemented by (Hopper and Turton, 2001) shows that ACO outperforms the other methods. Furthermore, the SPP problem is solved by a two phase algorithm where the learned order is mapped to the strip with the BLF heuristic. Thus, this demonstrates that ACO may be effectively customized for order learning problems and can be effectively coupled with non-standard heuristics (i.e., heuristics that are not incorporated within the solution construction). This shows that ACO can be further extended to test it with other problems consisting of similar characteristics. The implementation steps of ACO may be summarized as follows

- Choose an ACO variant
- Determine a learning model, i.e. pheromone trails
- Identify static heuristics which may be useful when constructing solutions
- Identify "ideal" parameter settings
  - Use previous studies as a guide and/or
  - Perform parametric optimization
- Identify convergence detection method if restarts are being used

2.3.2 Customizing Beam Search

To demonstrate beam search we consider an application from photo album layout (Atkins, 2008). The layout of photos effectively on web pages has received some attention. Usually, template-based solutions are provided where a user can place a

number of images. This can lead to problems when a user requires a large number of photos to be placed. The main issue is that a large number of alternative templates are plausible but a limited arbitrarily by the library providing them. The result is that the user has to settle for placements where photos may have an unexpected area and aspect ratio. A solution to this problem was proposed by Atkins (2008) using the Blocked Recursive Image Composition (BRIC) algorithm. This algorithm incrementally adds photos to an existing layout and selects placements which are closest to the area required by the user and maintain the aspect ratios of the photos. In the original version of the problem, photos must be spaced a minimum distance apart and they must have borders. We relax these constraints here since we observe that several layouts consist of very small photos with the borders and spacing overriding a significant amount of space occupied by the photos. This has the potential disadvantage that some photos may not be placed on the canvas. However, we argue that a placement where most photos fit their ideal areas is preferable to a placements where lot of small photos are placed where only the borders can be seen.

BRIC is implemented as a beam search. The basic idea here is that photos are added incrementally and at each stage a number of feasible layouts are constructed by testing all possible locations where the photo can be included. Of these, a limited number of layouts (partial solutions) are chosen based on a score of the current layout to progress with the search. The score (details provided later) is a measure of how close the the current photo areas are compared to the user specified photo areas. The current photos are required to fill the whole canvas and this results in a partial solution.

The above procedure effectively represents a beam search (also see C.1). Consider Algorithm 2. At line 5-18 we incrementally add photos to the layout. ProduceChildren(.) determines all possible feasible children and m of these are placed in the beam at line 16 of the algorithm (Reduce(.)). The additional steps at lines 8-14 in the algorithm are unused in this algorithm.

The following score is computed for the partial layout as follows. Given the suggested relative areas of the n photos $a_1, \ldots, a_n$, ideal area estimate for a photo i can be computed as:

$$\hat{a}_i = \frac{H \times W \times a_i}{\sum_{k=1}^{n} a_k} \tag{2.10}$$

where H and W are the height and width of the canvas, respectively. If the actual photo areas are represented as $o_1, \ldots, o_n$, then the score for each photo can be computed as:

$$s_i = min(o_i/\hat{a}_i, 1.0) \tag{2.11}$$

Therefore, this score imposes a penalty if the photo is smaller than its relative area but not if it is larger. The final score is calculated as the sum of the individual photo

scores:

$$S = \sum_{k=1}^{n} s_k \tag{2.12}$$

Atkins (2008), further modify the score in the following way. Firstly, they take into account any borders or spacing between photos. Secondly, further penalize a photo if its actual area is less than half the estimated area and thirdly, to impose a further penalty - a function of the number of photos that have a score less than 0.25. These last two modifications ensure that the layout does not consist of very small photos.

Adding a Learning Component

The above BRIC algorithm can be extended to include a learning component. ACO can be used for this purpose and over here we choose to implement in as a $\mathcal{MM}$AS in the hyper-cube framework, see (Thiruvady et al., 2009) for the general algorithm. We refer to it as BRIC-ACO from now onwards. Algorithm 2 is also a plausible framework for BRIC-ACO. From lines 5-18, photos are incrementally added to the layout. ProduceChildren(.) generates all feasible solutions by testing if a photo can be inserted at all possible locations on the tree. From amongst these, $\theta \times \mu$ solutions are selected using pheromones. Now the photos are sorted by score and the best θ are selected greedily to continue the search (line 16). Alternatively, the greedy selection and pheromone selection can be swapped and we test this version of the algorithm. In fact, the second alternative is more appropriate given that the score is already a proven estimate and the pheromone may initially eliminate promising solutions with high scores. Therefore, we stick with the second scheme of selection for the rest of this study. The order learning schemes in conjunction with BRIC are discussed in detail in Appendix C.2.

Experiments, Results and Discussion

Complete results, settings and details of the datasets are given in Appendix C.3. These results show that the BRIC algorithm with random orders is already very good. In fact, for all problems up to size 30, all the algorithms perform at similar levels, nearly always finding the optimal. However, for larger instances ($\geq$ 30 photos) BRIC-ACO is the best algorithm demonstrating the usefulness of the learning component. For these instances, even without learning, some randomness improves the solution quality. That is, BRIC-ACO-NL does not fix an order at the beginning of an iteration unlike BRIC. Therefore, the solutions in the beam are likely to be more varied resulting in the improvement over the plain BRIC algorithm.

The ACO component of Beam-ACO requires the same steps as outlined in Section 2.3.1. However, the steps involved with Beam search may be summarized as follows

- Determine selection scheme, i.e., pheromone selection followed by the estimate or vice versa
- Identify static heuristics which may be useful when constructing solutions
- Identify estimates, problem specific bounds or stochastic sampling
- Identify "ideal" parameter settings
  - Use previous studies as a guide and/or
  - Perform parametric optimization
- Determine greedy or probabilistic selection scheme for estimates

2.4 Conclusion

This section on learning metaheuristics lays the foundation for the first set of algorithms to be used in this thesis. ACO and Beam-ACO were the metaheuristics reviewed in depth. We discussed ACO first and compared to other metaheuristics, ACO is chosen since (1) it is proven, (2) more effective in practical settings than methods like BOA, (3) straightforward to integrate problem specific heuristics. Additionally, when applying ACO we have discussed two frequently used options in practice. The studies in the following chapters mainly use $\mathcal{MM}$AS but comparisons to ACS are made where it is deemed appropriate. The aim here is not to determine the most effective variant but to examine the effectiveness of the integrations with constraint programming. Thus, while improvements in the ACO components can be found, these improvements can be expected to extend into the respective hybrids without a major difference is the constraint programming component.

The second metaheuristic discussed was beam search. This approach combined with ACO - Beam-ACO - provides a way to enhance ACO in terms of flexibility and performance. Through an example we show how beam search differs from ACO and how it can more effectively exploit optimal regions. Apart from being an effective search method, beam search has the property that solutions are inter-dependent and are constructed in parallel. This allows for the exchange of information between the solutions while they are constructed as opposed to plain ACO. Often, identical solutions may be built by ACO (especially when pheromones converge). This can be avoided by using beam search to ensure that all solutions built in a single iteration

are unique. This aspect of beam search proves extremely useful when combined with constraint programming as will be seen in the next chapter.

We have demonstrated here that ACO and Beam-ACO can be effectively customized to problems based on order learning. This is particularly important in the context of this thesis since all three case studies considered can be represented by permutations and will thus be based on learning sequences or orders. Hence, customizing ACO/Beam-ACO for these problems is relatively straightforward.

Chapter 3

Constraint Programming

Constraint programming refers to the set of techniques for solving satisfaction and optimization problems (Marriott and Stuckey, 1998; Apt, 2003; Hentenryck, 1989). One of the first examples of a system using constraints was *sketch pad* (Sutherland, 1964). Since then, the development of CP started in the 1970's amongst in the AI community (Apt, 2003). They were the first to identify concepts such as *consistency* and specific algorithms aimed at solving them. The first CP systems originated in the 1980's (Borning, 1981; Steele, 1980) and were an extension of logic programming by incorporating the idea of *constraints*. This lead to the development of *constraint logic programming*. Independently of CP, search methods such as backtracking were already in use. Combining these with CP specific algorithms, CP systems became diverse, flexible and powerful.

At the heart of CP lie *variables*, *domains* and constraints. A problem is specified by a number of variables with associated domains and relations or constraints between the variables. The constraints are enforced by means of a *constraint solver* invoked by the program and all the constraints are placed in a *constraint store*. The solver analyses the constraints to enforce the implied restrictions. In doing so it provides at least two services to the program. First, it determines whether a new constraint is compatible with the already existing ones and reports success in the case that it is so. If not the solver reports failure. Secondly, given success, the domains of the variables are automatically checked and reduced to be consistent with the currently enforced constraints.

CP systems are very flexible in that they can deal with domain specific problems such as solving linear systems of equations or generic problems with constraints. In the case of linear systems, domain specific algorithms are usually applied. On the other hand, generic problems are usually modeled with CP specific algorithms designed to reduce the search space and possibly with CP specific search methods. CP also has a strong theoretical foundation and the ease of specifying CP programs and their flexibility make it an increasingly popular tool.

The program can interface with the solver in several ways. For example, a new restriction can be imposed or *posted* to the solver or a variable can explicitly be assigned a value. This is called a *labeling step*. The solver might reject such a labeling in case the value is not compatible with the constraints in the domain of the variable which will result in a state of failure. This is also the case with the constraints held in the solver. If consistent with the constraints, the post or labeling will possibly result in changes to the domains of the variables. The solver can also be queried by the program, e.g., to obtain the associated domain values of a variable. In this thesis, only CP over finite domains is considered and discussed in the next section.

3.1 Finite Domain CP

A branch of CP is finite domain constraint programming (FDCP[1]). This is a method for solving combinatorial problems (Marriott and Stuckey, 1998) where the domains associated with variables are finite sets (e.g., boolean or integer constraints). These types of problems are also referred to as constraint satisfaction problems (CSPs) if the aim is to find a solution that satisfies all the constraints of the problem rather than to optimize an objective function. Determining if a solution exists for this class of problems is generally $\mathcal{NP}$-hard. Therefore, exact/complete solution approaches, like integer programming, are generally very slow. Many real world problems such as scheduling, planning, packing, etc. involve choosing from a finite set of possibilities and can therefore be tackled with FDCP.

There are two phases in solving CSP. In the first phase the constraint model is defined and the initial constraints are posted. A constraint solver keeps track of the current variables and their domains in the constraint store for the problem. The second phase consists of assigning values to variables (*labeling*). Variable domains are updated by the constraint solver when an action (posting new constraints or labeling variables) is initiated. An action results in a new state for the solver, either success if the action is consistent with the constraints in the solver or failure if the action results in an invalid state. The two phases are described in detail in the following sections.

3.1.1 The CP model

In the first phase, the CP model is initialized with the variables and constraints. A *propagation* algorithm or *propagator* is invoked here to ensure consistency of the domains of the variables. These algorithms are called *solvers* and since solving a CSP

[1]The rest of this thesis will use CP and FDCP interchangeably unless otherwise specified.

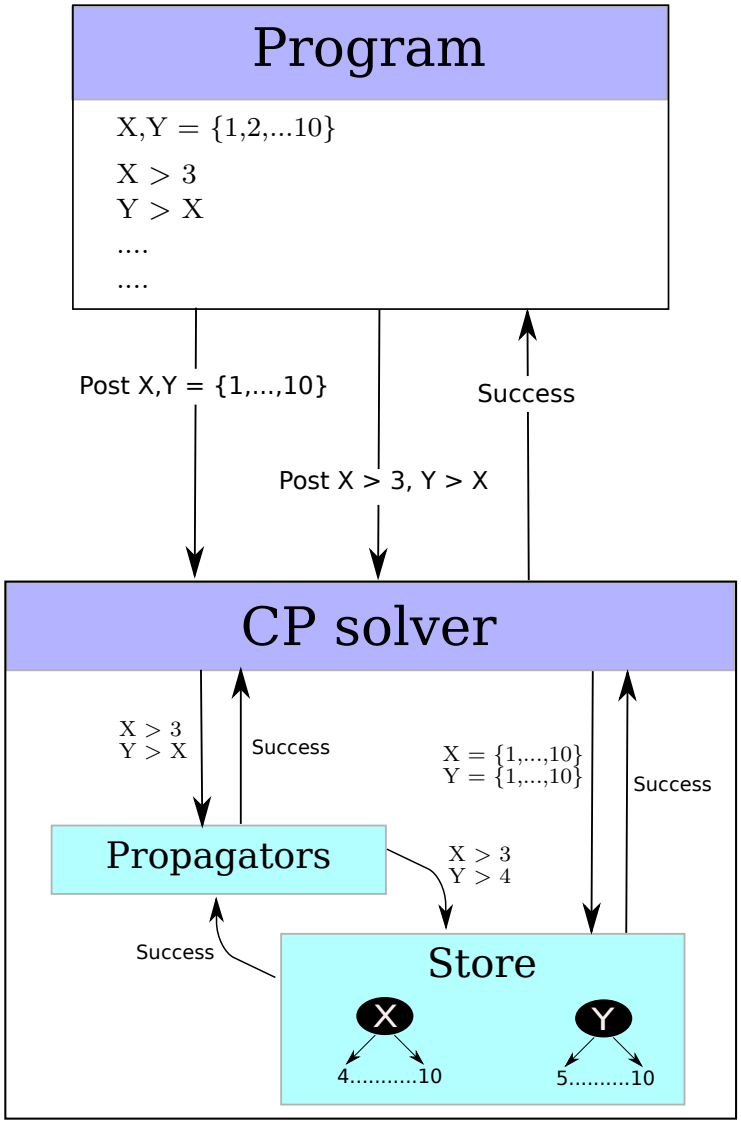


Figure 3.1: The interface between the program and the CP solver. Here, the program requests two variables $X, Y \in \{1, \ldots, 10\}$. This is set up by the solver and the state is placed in the store. The program now posts the constraints $X > 3$ and $Y > X$. These constraints are used by the propagators to infer the $Y > 4$ and this is applied to the variables in the store updating their domains. The solver returns success to the program to show that the actions led to consistent domains for he variables.

is $\mathcal{NP}$-hard they are necessarily incomplete. One class of solvers called consistency algorithms work by reducing the domains of the variables and thereby transforming the original CSP to an equivalent one. The new CSP consists of the same solution set as the old one and if it is determined that the new CSP is not satisfiable then the original CSP is also not satisfiable.

See Figure 3.1. Here, we see an example of a program that interfaces with the CP solver. First, two variables $X, Y \in \{1, \ldots, 10\}$ are defined by the program and posted to the solver. These variables and their associated domains are placed in the *store*. Since instantiating these variables was successful, the solver lets the program know that the variables were defined. Now further constraints may be posted to the solver which may invoke one or more propagators to reduce the domains of the variables or impose new constraints. In the example we see that $X > 3$ and $Y > X$ are posted to the solver. This is passed to the propagators which infer $Y > 4$. The constraints $X > 3$ and $Y > 4$ are applied to the store thereby reducing the domains of the the variables.

Algorithm 3 A simple consistency solver (Marriott & Stuckey, 1998)

INPUT: C, D
$\hat{D} = D$
// Node consistency
for $i = 1$ to $|C|$ **do**
  if $v(c_i) = 1$ **then**
    $x = v(c_i)$
    $\hat{D}(x) = \{d \in D(x) | D(x) \leftarrow d \text{ solves } c_i\}$
  end if
end for
// Arc consistency
repeat
  $K = \hat{D}$
  for $i = 1$ to $|C|$ **do**
    if $v(c_i) = 2$ **then**
      $x_1 = v_1(c_i)$
      $x_2 = v_2(c_i)$
      $\hat{D}(x_1) = \{d_1 \in \hat{D}(x_1) | \text{ for some } d_2 \in \hat{D}(x_2), x_1 \leftarrow d_1 \wedge x_2 \leftarrow d_2, \text{ solves } c_i\}$
      $\hat{D}(x_2) = \{d_2 \in \hat{D}(x_2) | \text{ for some } d_1 \in \hat{D}(x_1), x_1 \leftarrow d_1 \wedge x_2 \leftarrow d_2, \text{ solves } c_i\}$
    end if
  end for
until $K = \hat{D}$
if $\hat{D} = \mathit{invalid\ domain}$ **then**
  return false
else if $\hat{D} = \mathit{valuation}$ **then**
  return true
else
  return unknown
end if

The solver works as follows. Each primitive constraint (c_i) is considered and given the domain of a variable, the domains of all other variables are reduced or made "consistent". This process is repeated for all constraints and could possibly result in a *failure* where the CSP can not be solved or a *valuation* where each variable is assigned to a single value. A simple solver can be built from a two step algorithm that employs *node* consistency followed by *arc* consistency (see Algorithm 3). In node consistency, all constraints involving only a single variable ($v(c_i) = 1$) are checked to ensure that their domain values satisfy the constraint (lines $4-9$). C are the set of constraints and D are the domains of the variables. At the end of the first part (line 9), $\hat{D}$ has the reduced domain values. This if followed by arc consistency (lines $11-21$) which is applied to all constraints with two variables ($v(c_i) = 2$) and require that for all domain values for the first variable, some domain value of the second variable will result in a feasible solution to the constraint (line 17). Consistency is also enforced from the second variable to the first (line 18). Again $\hat{D}$ results in the set of pruned domains.

For constraints with more than two variables, methods such as *hyper-arc consistency* can be employed, however, the problem of identifying if a primitive constraint is hyper-arc consistent is $\mathcal{NP}$-hard (Marriott and Stuckey, 1998). Fortunately, in the case of arithmetic constraints, *bounds* consistency can be used for more than two variables. Arithmetic CSPs are the most common type CSP, and often, those that are not can be converted to an integer CSP by requiring integral domains. A primitive constraint c is bounds consistent if (Marriott and Stuckey, 1998)

- $\forall x \in \{x_1, \ldots, x_n\} : \forall y \neq x, \{min(d_{y_j}) \leq r_j \leq max(d_{y_j})\} \wedge \{x \leftarrow min(d_x), y_1 \leftarrow r_1, \ldots, y_n \leftarrow r_n\}$ solves c and

- $\forall x \in \{x_1, \ldots, x_n\} : \forall y \neq x, \{min(\hat{d}_{y_j}) \leq \hat{r}_j \leq max(\hat{d}_{y_j})\} \wedge \{x \leftarrow max(d_x), y_1 \leftarrow \hat{r}_1, \ldots, y_n \leftarrow \hat{r}_n\}$ solves c

The first point specifies that there exist real values r_j in the range of the values of each variable y_j such that this variable can be bound to this value, and x being bound to its minimum domain value, results in a solution to the constraint. The second point is similar to the first except that x being bound to its largest domain value results in a solution to the problem. Alternatively, either $min(x_j)$ or $max(x_j)$ and all constraints can be satisfied by assigning values to all other y_i in their respective domains. The arithmetic CSP is then said to be bounds consistent if all the primitive constraints are bounds consistent. For a more complete description of CSP solvers refer to (Marriott and Stuckey, 1998).

3.1.2 Labeling

Once the CP model is defined, a propagator is triggered which will apply the consistency algorithms. This results in the updating of the domains of the variables and the solver is now in an initial *state*. Here, we may post a constraint to assign a value to a variable which is also called *labeling*. If all the variables are bound at the initial state then there is no labeling phase required and a solution to the problem is returned. Otherwise, variables are labeled while the solver does not discover a state of failure. Typically, a backtracking search is the simplest way this can be implemented. A variable is bound to a value, the propagator is invoked, and then another variable is bound. This process continues until a solution is found or the solver state results in failure. In the latter case the algorithm would backtrack to the previous variable and make a different choice. This process is repeated until a solution is found or no solution exists.

The choice of which variable to label next is also important. Often in applications, the variable with the smallest domain is an obvious choice. However, for some problems labeling variables based on other choices is more effective.[2]

In the case of optimization problems (e.g. integer optimization), algorithms from linear/integer programming can be used in combination with CP solvers for an efficient search. An example is a branch-and-bound search strategy. Let *opt* be an algorithm which is an optimizer for real numbers. Let *opt* find a solution to the original problem with the integrality constraints relaxed. If the resulting solution is integral then the solution may be returned without any further search. Otherwise, we recursively pick a non-integral variable (say $x = v$) and search two separate problems ($x \geq \lceil d \rceil \wedge x \geq \lfloor d \rfloor \wedge C$) where C are the constraints. While searching, bounds obtained from the relaxed version of the problem can be used to prematurely stop exploring a path of the search tree.

Consider the previous example from Figure 3.1. The program posts X, Y $\in \{1, \ldots, 10\}$ and two constraints $X > 3$ and $Y > X$. Now we attempt to label X. The labeling procedure is shown in Figure 3.2. Observing successive labelings from left to right, $X = 3$ is first posted. This is not consistent with the domain of X and is rejected by the solver. The program then requests that the domains be reset and then attempts $X = 5$ which succeeds. The propagators infer $Y > 5$ and the domain of Y is updated accordingly.

[2]For example, consider the N-queens problem which is a satisfaction problem where N queens have to be placed on a an $N \times N$ chess board such that no two queens can capture each other. Here a 'middle' variable is a better choice to instantiate first (Marriott and Stuckey, 1998).

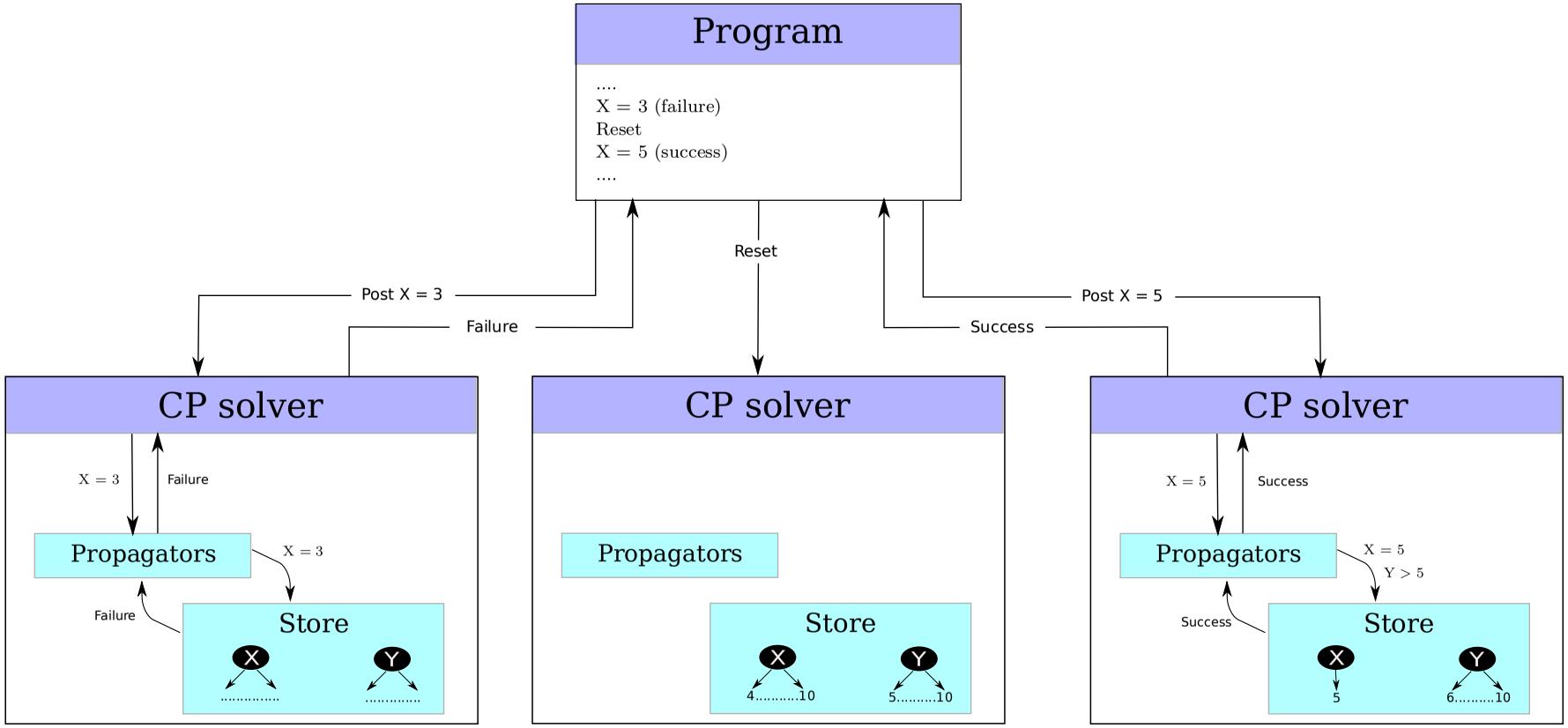


Figure 3.2: A simple example of specifying constraints and labeling. In the first post, the domains of two variables X and Y are set up. The second post specifies the constraints which proceeds to prune the domains of both variables. Then a labeling is attempted by setting $X = 3$ which is inconsistent with the domains. This results in failure. After resetting the solver state labeling $X = 5$ is posted which results in X being bound.

3.1.3 High-Level constraints

Several constraints are considered as high-level in that they consist of a number similar primitive constraints. When specifying a CP model for a problem, it is generally useful to use high-level constraints (also called Global constraints) if the model permits it. This is because specialised algorithms for the high-level constraint may be available which permit additional inference. Furthermore, explicitly specifying each constraint can be tedious.

For example, consider the following constraint: $distinct(X), X = \{x_1, \ldots, x_n\}$. This constraint is equivalent to $x_i \neq x_j, i \neq j$. However, pruning the domains of the variables can be mapped to the maximal bipartite matching problem (discussed in Chapter 1) which can be tackled efficiently with flow algorithms (Milano, Ottosson, Refalo and Thorsteinsson, 2001). Furthermore, explicitly specifying these constraints requires $\frac{n\times(n-1)}{2}$ posts.

Therefore, for the applications in the following chapters, we make use of the following high-level constraints, possibly using more than one for each problem:

- $distinct(X) : \forall x_i \neq x_j, i \neq j, X = \{x_1, \ldots, x_n\}$. As discussed earlier, this constraint specifies that all variables must be labeled with different values.

- $count(X, d, m) : num(x_i = d, i \in 1, \ldots, n) = m$. This constraint specifies that the number of occurrences of d amongst the variables is exactly equal to m.

- $cumulative(S, D, R, l) : \forall t \sum_{i|s_i \leq t \leq s_i + d_i - 1} r_i \leq l$ (Apt, 2003). This constraint was designed for scheduling problems. For m jobs, the constraint specifies determines the start times ($S = \{s_1, \ldots, s_m\}$) given the durations ($D = \{d_1, \ldots, d_m\}$) and resource consumptions ($R = \{r_1, \ldots, r_m\}$) of the tasks. l is the maximum resource available at a time point t. The constraint specifies that the resource consumption by all tasks that overlap at any time point must be no more than l.

- $among(X, s, l, h) : l \leq num\{j \in \{1, \ldots, n\} \mid x_j = s\} \leq h$. This constraint requires that the number of occurrences of $x_j = s$ is less than h and greater than l.

- $sequence(X, s, q, l, h)$: $\bigwedge_{i=0}^{|x|-q} among(\langle x_i, \ldots, x_{i+q-1}\rangle, s, l, h)$ (Gecode, 2010). Using the $among(.)$ constraint, the sequence constraint enforces this constraint for every subsequence of X of size q.

3.1.4 Choice of CP Solvers

Constraint solvers for combinatorial problems are commercially and freely available (ILOG, 2007; Carlsson, Ottosson and Carlson, 1997; Gecode, 2010). In order to settle on an appropriate solver we considered the following aspects:

- Up-to-date: the solvers propagation algorithms are state-of-the-art with strong propagation
- Reliable: has been tested by other users without problems
- High-level constraints: includes an implementation for commonly occurring high-level constraints (e.g. *distinct*) and include specialized high-level constraints that may be needed for specific applications (e.g. *sequence* for car sequencing)
- Copying solver state: this is critical to the performance of CP-Beam-ACO (see next chapter) and needs to be implemented efficiently
- Interface: can interface to the solver libraries in a straightforward manner
- Statistics: can obtain statistics over propagation steps, execution times, labelings, etc.

Given the above criteria, the CLPFD library of Sicstus and Gecode were used in this thesis. The ILOG solver was not considered since all the algorithms tested here involve a stochastic component and proving the effectiveness of these algorithms is achieved through a large number of runs for each algorithm on each instance. To obtain these results on many instances requires that these runs be executed in parallel on a cluster or several clusters of computers. This would involve a large number of commercial licenses which were infeasible to obtain.

Initially, Sicstus was the preferred solver since the original work on combining ACO and CP (Meyer and Ernst, 2004) used this solver effectively applying the high level scheduling constraints.[3] However, this solver has the drawback copying a complete solver state efficiently was not possible. This provided an impediment to the main algorithm suggested in this thesis (CP-Beam-ACO), i.e., constructing solutions in parallel and moving between parts of the search tree is not possible. To achieve this, effective copying of solver sates is absolutely necessary and Gecode provided this mechanism in addition to making available all the relevant high level constraints needed that are described in the previous section. Therefore, all the experiments and associated results in this thesis were obtained with the Gecode CP solver.

[3] *serialized*$(\cdot)$ in sicstus equivalent to *cumulative*$(\cdot)$.

3.2 Conclusion

Constraint programming is an effective method to tackle tightly constrained problems. In particular, finite domain constraints can be used to effectively with real world problems, since a large number of these problems consist of constraints with discrete components. This chapter provided on overview of how a problem can be solved with CP using finite domains including modelling the problem. We have also provided some insight into the inner workings of CP by describing some of the consistency algorithms briefly. High-level algorithms were also discussed.

An important aspect of CP is that it is declarative, allowing a high-level description of the problem. Furthermore, a problem that is modeled using high-level constraints amongst others, can achieve significantly improved results with respect to feasibility. A possible drawback, however, is that for large search spaces CP on its own can require far too much effort in its search component. Given these aspects, an effective integration can make use of CP to find feasibility on tightly constrained problems. Such integrations are discussed in the next chapter.

Chapter 4

CP-Beam-ACO

Often, COPs may have characteristics which are hard to solve by a single class of techniques. Thus, combining methods from different classes provides a mechanism to be able to deal with these problems. Given the potential of hybrid methods, they have been given a lot of attention recently and the growing number of studies with hybrid methods suggest that they will be an effective and frequently utilized technique in the future (Blum, Puchinger, Raidl and Roli, 2010; Puchinger and Raidl, 2005).

Developing a hybrid algorithm may not be straightforward. The combinations can be implemented in several ways and designing the "best" hybrid can be difficult. In this chapter we discuss several studies which show how hybrids may be constructed and we also discuss the frameworks into which hybrid methods have been classified.

The focus in this thesis is on COPs with non-trivial hard constraints and we aim to tackle these problems with ACO and CP and their integration. Thus we review hybrids of metahuristics and exact methods. Additionally, we suggest a further hybrid with beam search, motivating the need to discuss hybrid metahuristics.

As we pointed out in Chapter 1, metaheuristics deal with non-trivial hard constraints poorly. The approaches usually build solutions to these problems by relaxing constraints, repairing solutions which violate constraints, or using multi-phase techniques. Relaxation or penalty methods build solutions which are allowed to violate constraints. Typically, the original objective is modified to include the cost of the violations where a solution to the problem is built by mininmizing a function of the violations. Repair methods also first build infeasible solutions. These solutions are examined and modified or re-built to obtain feasible solutions. Finally, multi-phase methods build solutions in two phases. The first is to bias the solution construction into feasible regions until feasible solutions are found and the second phase to find optimal areas in the feasible regions found.

In all the cases discussed above it is often a tedious task to try to build feasible solutions. Furthermore, the methods developed are problem specific and may not easily extend to other problem types. The obvious advantages of using a CP-based hybrid is that it is problem independent. CP allows the modeller to specify constraints at a high-level independent of the specific problem. Hence, CP provides a relatively simple and effective way to model the constraint of a problem and the resulting framework may be applied to several problem types. This CP-ACO algorithm is the basis of this thesis and is reviewed in detail followed by a discussion of the CP-Beam-ACO hybrid.

4.1 Hybrids of Metaheuristics and Exact Methods

Puchinger & Raidl (2005) classify hybrid techniques into collaborative and integrative combinations. The aim of these hybrids is to either improve the run-time of the algorithm when solving a problem or improve the solutions that are found given the same time-frame.

The first combination is one where both algorithms exchange information but execute independent of each other in sequence (Clements, Crawford, Joslin, Nemhauser, Puttlitz and Savelsbergh, 1997; Applegate, Bixby, Chvátal and Cook, 1998; Plateau, Tachat and Tolla, 2002; Klau et al., 2004) or in parallel (Talukdar et al., 1998; Denzinger and Offermann, 1999). In sequential integrations, the metaheuristic or exact method is used as an initialization or pre-processing to provide a point at which the other method may be used. Clements *et al.* (1997) tackle a production-line scheduling problem. Initially, they use squeaky wheel optimization (Joslin and Clements, 1998) to generate feasible solutions and also include random component to generate diverse solutions. This is followed by a second phase where a column generation approach is used to improve the solutions. Applegate *et al.* (1998) use iterated local search first to generate a number of solutions to the TSP. Given these solutions, the TSP is solved to optimality on the edge set of the solutions found, hence leading to improved solutions compared to iterated local search on its own. Klau *et al.* (2004) tackle the prize-collection steiner tree problem in a similar fashion by first using a memetic algorithm followed by integer programming. Plateau *et al.* (2002) make use of an interior point algorithm to find initial solutions to a multi-constrained knapsack problem. This is used to provide the initial population for a path-relinking algorithm. Puchinger & Raidl (2005) also discuss other approaches and we refer the reader to this survey to get further details.

Integrations in parallel are less common but have also shown to be useful. Talukdar *et al.* (1998) present a framework for such integrations as follows. They consider an agent-based approach where each agent is an optimization algorithm. These agents work on shared memory which consist of solutions to the problem, a relaxation (super class) or subclass. Their actions can result in adding, deleting or altering solutions to each class of the problem. Denzinger and Offerman (1999) also present a similar approach where they show they effectiveness of genetic algorithm and branch & bound search on a job-shop scheduling problem.

The second type of combination is when one of the algorithms (a slave) is built into the other (master). Examples of exact techniques incorporated into metaheuristics are (Chu and Beasley, 1998; Raidl, 1998; Meyer and Ernst, 2004). Chu & Beasley (1998) and Raild (1998) tackle the multiconstrained knapsack problem by solving relaxed problems within a genetic algorithm. Here, the relaxed problems are solved exactly and they are used to guide a neighbourhood search, mutation, local improvement, etc. or combinations of these. Exact methods may also be used to explore large neighbourhoods exactly (Thompson and Psaraftis, 1993; Burke, Cowling and Keuthen, 2001; Congram, 2000; Puchinger, Raidl and Koller, 2004). Burke *et al.* (2001), for example, tackle the ATSP with a local and variable neighbourhood search and exhaustively explore the neighbourhood in the local search component. Meyer & Ernst (2004) examine the CP with ACO integration which is discussed in detail later.

Studies incorporating metaheuristics into exact algorithms include (Woodruff, 1999; Filho and Lorena, 2000; French, Robinson and M.Wilson, 2001; Fischetti and Lodi, 2003; Kostikas and Fragakis, 2004; Puchinger and Raidl, 2004; Danna, Rothberg and Pape, n.d.). Woodruff et al. (1999) use tabu search within a branch & bound search to obtain bounds or an incumbent solution.[1] Filho *et al.* (2000) and Puchinger and Raidl (2004) use metaheuristics and heuristics within branch-and-cut (dynamic separation of cutting planes) and branch-and-price (pricing of columns) algorithms. Guiding branch & bound searches with metaheuristics has also been attempted, for example, by French *et al.* (2001) and Kostikas *et al.* (2004). They use genetic algorithms and genetic programming, respectively, to guide the node selection in the branch & bound trees. Finally, Puchinger & Raidl (2004) discuss approaches where a neighbourhood search around an incumbent solution is attempted before the traditional node selection strategy of branch & bound. The initial neighbourhood search is itself done with a branch & bound search as opposed to a metaheuristic, however, this can be viewed as a heuristic approach. Fischetti & Lodi (2003) and Danna *et al.* (2005) are examples of such approaches.

[1] Current best feasible solution

4.2 Hybrid Metaheuristics

There are various categorizations of metaheuristics. Here, following (Blum et al., 2010) we broadly discuss metaheuristics incorporated within other metaheuristics, metaheuristics integrated with tree search methods, metaheuristics and CP and other hybrids.

Metaheuristics are often combined with heuristics and possibly combined with other metaheuristics. In this class of algorithms, typically, local search algorithms may be used within a metaheuristic framework (see for example (Krasnogor and Smith, 2005)). Local search methods are essentially intensification algorithms and metaheuristics are effective at diversification. Therefore, the resulting hybrids make use of the complementary advantages of both methods leading to more effective searches where the metaheuristics explore large areas of the search space and the local search identifies the best solutions in those areas quickly.

Integrations with tree search methods can be effective where metaheuristics are constructive in nature. As we saw in Chapter 2, ACO can be viewed as a tree search method and a solution can be thought of as a path from the root to a leaf. This leads to integrations with branch & bound methods or derivatives of branch & bound methods (see for example (Blum, 2005)). Furthermore, if efficient integer programming or CP formulations can be devised, hybrids where the metaheuristic makes use of the exact method can be effective by using powerful solvers to guide the search. Metaheuristics may also be used to guide a search within an exact method. For example, (Rothberg, 2007) suggested a branch & cut algorithm which makes use of an evolutionary algorithm at regular intervals as a heuristic to improve solution quality.

Integrations with metaheuristics and CP include (Meyer and Ernst, 2004; Backer, Furnon, Shaw, Kilby and Prosser, 2000; Shaw, Backer and Furnon, 2002; Focacci, Laburthe and Lodi, 2001). These integrations can be approached in different ways as follows. The first approach is collaborative, i.e., first execute one algorithm and use the solutions generated to guide the second algorithm. A second method may use CP to effectively explore the neighbourhood of a solution, which is typically useful with local search. Thirdly, the metaheuristic's search may be guided using CP, leading to feasible regions of the search space. Lastly, during the CP search a metaheuristic may serve to improve a solution.

Another category of hybrids include those which relax some of the problem constraints and then make use of other methods to efficient solve the relaxed problem (Vasquez and Vimont, 2005; Raidl and Feltl, 2004). This is similar to integer programming methods, which when integrality constraints are relaxed, can obtain bounds via the linear programming solution for their search. Raidl and Feltl (2004)

make use of such a method to obtain an initial population of integral solutions by using a linear programming relaxation for the generalized assignment problem.

Finally, metaheuristic integrations with dynamic programming have proved effective (Blum and Blesa, 2008). For example, when large neighbourhoods (exponential in size) have to be dealt with, dynamic programming methods can sometimes explore such search spaces in polynomial time. The study by Blum & Blesa (2008) tackles the k-cardinality tree problem. Here, a metaheuristic is used to generate a group of solution from which dynamic programming is used to determine the best solution amongst these.

We made the point earlier that metaheuristic-CP integrations can be effective hybrids. This is particularly true for COPs with non-trivial hard constraints. How such hybrids can be developed are discussed next and future chapters show how they can be customized for three separate problems.

4.3 CP-ACO

Metaheuristics deal with hard constraints poorly and most traditional methods tend to be problem specific. Since many real world problems are subject to such constraints, metaheuristics can benefit from using CP to effectively deal with non-trivial hard constraints to build feasible solutions to problems. One of the first examples of such an integration was to use CP with a genetic algorithm (Barnier and Brisset, 1998). CP was integrated with the genetic algorithm where new solutions are generated. For example, new individuals (solutions) that are generated after applying crossover are examined by the CP solver for feasibility. While this proves effective, the resulting hybrid possibly requires a large computational effort when determining feasibility. In fact, this study imposes a time limit on propagation after which solutions are regarded as having low fitness. The alternative of constructive metaheuristics provide a straightforward way to deal with CP and can often determine early on if a solution, when completed, will be feasible. Meyer & Ernst (2004) showed that this can be effectively done with ACO and CP and the hybrid was particularly effective on a tightly constrained COP.

The problem (Meyer and Ernst, 2004) tackled was a single machine job scheduling with sequence dependent setup times.[2] The problem consists of non-trivial hard constraints in release times and due times associated with each job. The objective was to minimize *makespan* or the completion time of the last job in the sequence. It can also be viewed as minimizing the cumulative setup times while satisfying the hard constraints. Typically, ACO use heuristics such as earliest due date to find feasible solutions followed by shortest setup times to find optimal solutions. CP on

[2]See Chapter 5.1 for the formal definition of the problem.

Algorithm 4 Solution construction for ACO with CP

```
1: INPUT: A problem instance
2: π = ∅
3: failed = false
4: while i ≤ n ∧ failed = false do
5:    feasible = false
6:    π̄ = π, D = domain(π_i)
7:    // single level backtracking
8:    while feasible = false ∧ D ≠ ∅ do
9:       j = select_component(D, T)
10:      feasible = post(π̄_i = j)
11:      D ← D \ j
12:   end while
13:   if feasible = true then
14:      π_i = π̄_i
15:   else
16:      failed = true
17:   end if
18: end while
19: OUTPUT: a feasible solution π or null
```

the other hand only allows constructing feasible solutions and using CP within ACO allows the algorithm to focus on optimizing the schedules. The results show that for smaller or less tightly constrained problems, ACO with heuristics are sufficient. However, when the problem sizes get larger and/or more tightly constrained then CP-ACO integration is the preferred option.

The study above is an example of the CP-ACO integration for optimization problems with constraints. CSPs have also been explored with a similar integration recently (Khichane, Albert and Solnon, 2008). They considered car sequencing and a very similar algorithm to above CP-ACO hybrid was proposed. This is a pure satisfaction problem, however, and Kichane *et al.* (2008) showed that ACO can be used to improve a pure CP approach to more effectively identify feasible sequences.

Meyer & Ernst (2004) showed how ACO can be easily extended to use CP in its solution construction. The algorithm is shown in Algorithm 4. Lines $4-18$ in the algorithm label each variable sequentially. At each variable, single level backtracking is used (lines $8-12$) to determine a value that is consistent with the solver. The domain value d is initially picked using pheromones (see the previous Chapter) and tested by the solver for feasibility (lines $9, 10$). The domain value tested is removed from the domain of the variable in line 9. If feasible, the variable is labeled otherwise the procedure continues until a valid value is found or the domain of the variable becomes empty ($D(i) \neq \emptyset$). If a value is found then the process continues to the next variable and, otherwise, terminates with infeasibility (lines $13-17$).

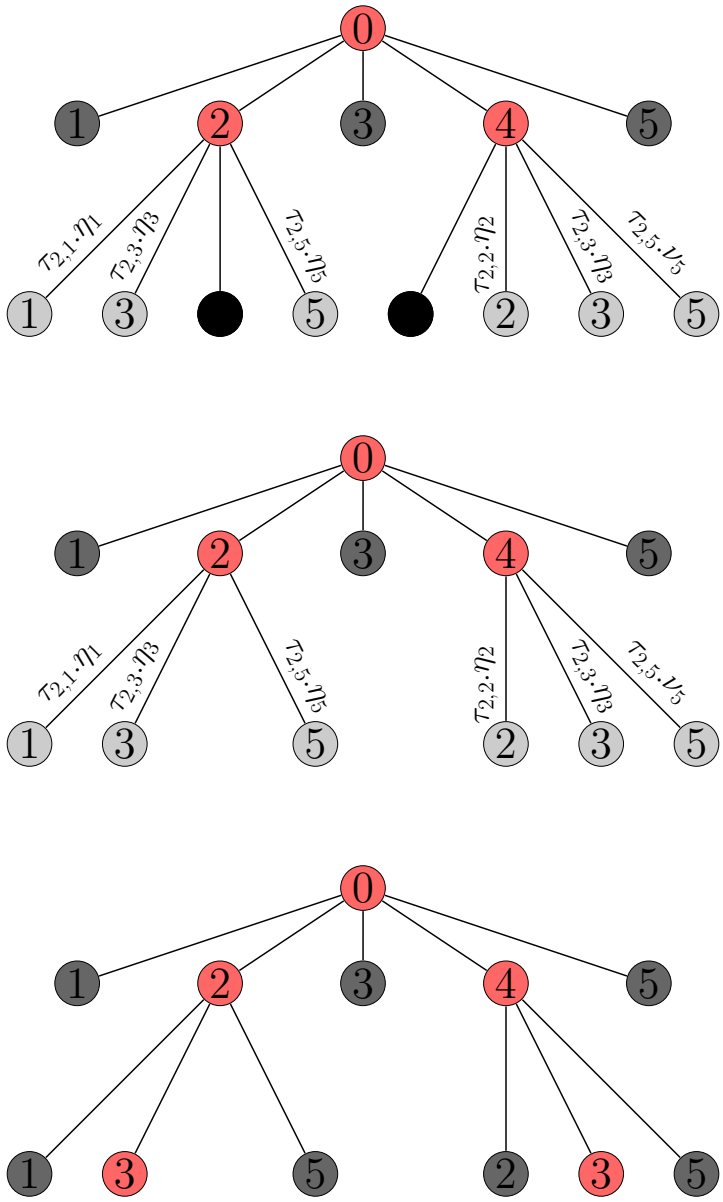


Figure 4.1: CP-ACO as a tree search. Node 0 is the root of the tree where the solution construction starts. The first variable is represented at level 1. In this example, two ants are constructing solutions independently and have chosen $x_1 = 2, \pi_1 = \{2, \ldots\}$ and $x_1 = 4, \pi_2 = \{4, \ldots\}$, where the index represents a variable. The figure on top shows the pheromones and heuristics factors that are considered by each ant when attempting to select the next component. Additionally, CP eliminates two nodes (black-filled nodes), one of which was originally chosen by ACO. Given this information, the first ant chooses $x_2 = 3, \pi_1 = \{2, 3, \ldots\}$ and the second ant selects $x_2 = 3, \pi_2 = \{4, 3, \ldots\}$.

Figure 4.1 shows CP-ACO as a tree search. Comparing this to ACO in the previous chapter (Figure 2.1), we see that solutions that cannot lead to feasible solutions are eliminated as potential candidates. For example, the solution with $\pi_1 = 2$ cannot have $\pi_2 = 3$. Thus, only feasible solutions are selected. This example shows how the solution constructions of ACO and CP-ACO are potentially very different for tightly constrained problems.

We have discussed the point earlier that finite domain CP solvers are slow. Often, satisfying the constraints to obtain a solution is itself an $\mathcal{NP}$-Hard problem. Therefore, an inherent disadvantage of CP-ACO is that the algorithms require large run-times to carry out the same number of solution constructions as a basic ACO algorithm. This was the case as reported by the previous two studies and motivates the need for faster solvers. The algorithms developed in this thesis aim to show that this is possible by parallelizing the solution construction via beam search.

4.4 Integrating CP with Beam-ACO

CP-ACO is computationally demanding due to CP propagation. While stronger CP models will lead to finding feasibility faster, there is a price to pay for this advantage. CP propagation costs are dependent on the CP model, i.e. stronger CP models require larger computational effort. In fact, the CP model suggested by Meyer & Ernst (2004) was complex leading to large execution times for calls to the CP solver. Thus, repeated calls to the solver for the same propagation is inefficient. Unfortunately, due to ACOs solution construction scheme, CP-ACO inherits this inefficiency. The aim is to improve CP-ACO due to the reasons outlined above. The algorithm may be made more efficient by (1) avoiding duplication by eliminating repeating CP solver calls or (2) by directing the search effort towards "new" areas of the search space.

The inefficiency of ACO to be addressed is that there is repeated path generation where the same solution components are selected during construction. This is significantly more so when the pheromones have converged. If the CP solver is invoked for each newly added component, for repeating solutions, we have a large number of calls which are effectively carrying out the same work. This can be avoided in two ways: (1) make sure that all solutions constructed in an iteration are not repetitive and (2) solutions constructed between iterations are not repeated.

We focus on the first point in this thesis. To achieve this, all solutions within an iteration must be constructed in parallel. Beam search is one such algorithm that achieves this as part of the construction process. We have discussed how beam search and ACO may be combined effectively leading to Beam-ACO. Thus, by integrating

Algorithm 5 Probabilistic Beam Search with CP

1: INPUT: $(\theta, \mu, \mathcal{T})$
2: $B_0 = \{\pi_1 = (), \ldots, \pi_\theta = ()\}$
3: $t = 0$
4: **while** $t < n$ **and** $|B_t| > 0$ **do**
5: **for** $i \in B_t$ **do**
6: $k \leftarrow 0$, $D = \text{domain}(\pi^i_{t+1})$
7: **while** $k < \mu \wedge D \neq \emptyset$ **do**
8: // $\overline{\pi}$ is a copy of π^i including a copy of its solver state
9: $\overline{\pi} = \pi^i$
10: $j = \text{select\_component}(D, \mathcal{T})$
11: *feasible* $= \text{post}(\overline{\pi}_{t+1} \leftarrow j)$
12: **if** (*feasible*) **then** $B_{t+1} = B_{t+1} \cup \overline{\pi}$
13: $k \leftarrow k + 1$, $D = D \setminus j$
14: **end while**
15: **end for**
16: $B_{t+1} = \mathsf{Reduce}(B_{t+1}, \theta)$
17: $t \leftarrow t + 1$
18: **end while**
19: OUTPUT: $\text{argmax}\{f(\pi) \mid \pi \in B_{n-1}\}$

CP into Beam-ACO we have an algorithm that avoids re-constructing parts of the search space and achieves the desired result of efficiently handling CP.

Beam search can be extended to include CP (see Algorithm 5). Here, at the first stage of extending a partial solution, the choice obtained from sampling the pheromones is posted to the solver (line 11). If the choice is feasible then this partial solution is placed in the beam. Otherwise the selection is removed from the plausible domain values (line 13) and the procedure continues.

This procedure will only be successful if the copying of solver states is relatively simple. GECODE (Gecode, 2010) provides an effective way to handle this. The major CP-related advantage of this method is that propagation is only ever done for new solutions and re-propagation is never done within an iteration. Another advantage is that if certain partial solutions become infeasible, then other feasible partial solutions may be placed in the beam. In CP-ACO infeasible solutions are lost and no new solutions are examined instead.

See Figure 4.2. Compared to beam search, this figure shows that at the initial stage of selection, we first select using pheromones (τ) and heuristic information (η). Additionally, some solution extensions may be discarded based on querying the CP solver. Given the reduced candidates, we now select from them using the estimates (ϕ). As this example shows, the search may jump from one (currently feasible) part of the tree to another, thereby focusing on promising areas of the search space.

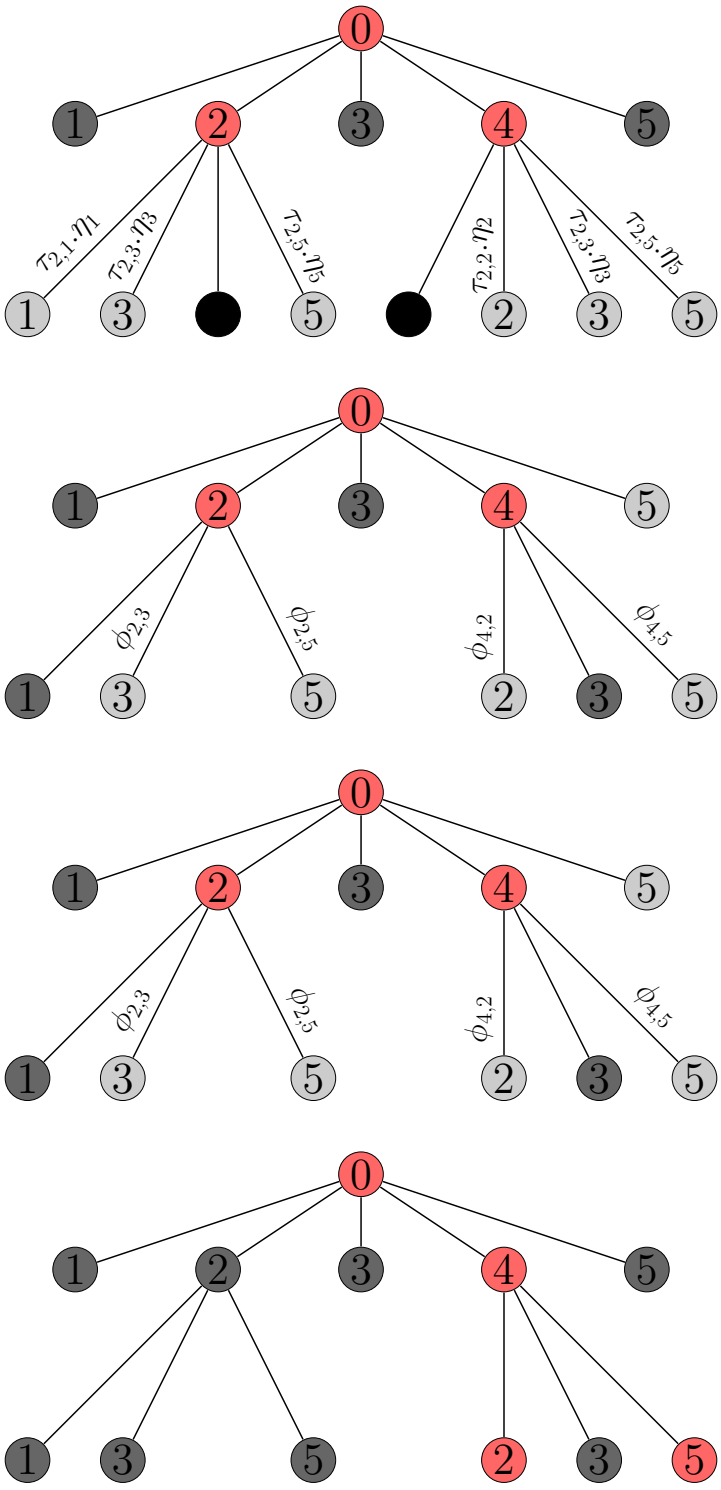


Figure 4.2: Partial construction of a solution using CP-Beam-$\mathcal{MM}$AS. Here, two ants are constructing solutions in parallel and have chosen $x_1 = 2, \pi_1 = \{2, \ldots\}$ and $x_1 = 4, \pi_2 = \{4, \ldots\}$, where the index represents a variable.The search proceeds similar to the Beam-$\mathcal{MM}$AS method except that infeasible solutions are ruled out by CP at the first stage (black-filled nodes). As this example shows, different solutions may be obtained compared with Beam-$\mathcal{MM}$AS as a result. The partial solutions generated have the following components: $x_2 = 3, \pi_1 = \{4, 2, \ldots\}$ and the second ant selects $x_2 = 3, \pi_2 = \{4, 5, \ldots\}$.

The different stages of selection do not necessarily have to be executed in the order suggested above. We have used pheromone-based selection first to make use of learnt information. This could have been replaced by using the estimate to determine the initial candidates. However, computing the estimates is often expensive compared to the constant time look-up of pheromone information. Thus, it is more efficient to select initial candidates based on the pheromone information. Additionally, eliminating infeasible solutions is desirable and hence CP is used early on so that the largest possible candidate set is available to sample from. It does not seem useful to use CP at the second stage given that the choices made to this point may already be infeasible. Finally, as we have pointed out earlier, computing the estimates is often expensive and is left to the last stage of selection. Furthermore, using the pheromones at the second stage is often ineffective as we only obtain information about the next level of selection. Pheromones favour certain paths and are guided by the locally best selection which may not be an effective choice across all levels of the search tree. So choosing to move from one part of the search tree to another based on the pheromones is unlikely to be effective.

4.4.1 Characteristics of CP-Beam-ACO

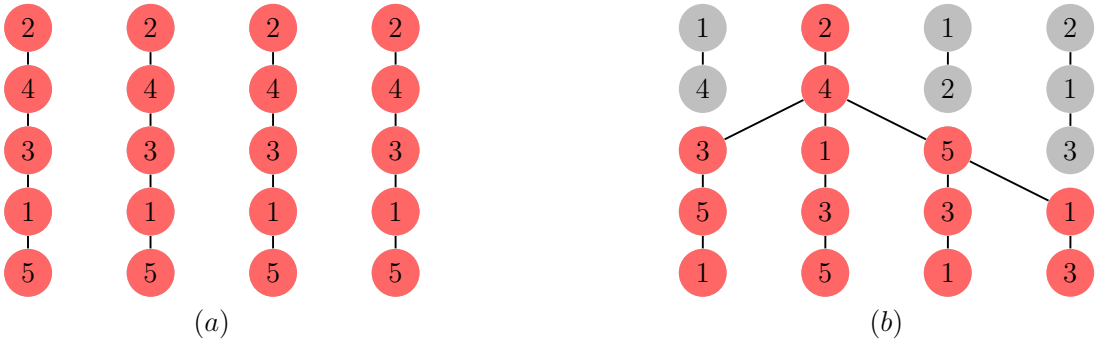


Figure 4.3: This figure demonstrates how the solution construction may proceed for a when constructing a permutation of five elements. (a) shows that CP-ACO might generate a large number of identical solutions. (b) shows that CP-Beam-ACO will force the solutions to be unique.

Beam search integrated with CP-ACO proves to be advantageous in several aspects. The main advantage of CP-Beam-ACO is that it forces the solution construction to diversify. Hence, solutions are never repeated within an iteration but the search is focused around different (optimal) regions. This aspect can be seen in Figure 4.3 which shows that CP-ACO can repeatedly construct the same solution whereas CP-Beam-ACO will force the construction to examine new solutions.

A second advantage is that CP-Beam-ACO examines a larger number of solutions compared to CP-ACO in the same time-frame. The reason for this is that CP-Beam-ACO examines many more solutions near the end of the search tree where invoking

the CP solver is relatively quick. Often, it is at the end of the search tree that solutions may be found to be infeasible and here CP-Beam-ACO creates many more solutions and hence a lot of quick labelings.

The third advantage of CP-ACO is demonstrated in Figure 4.4. We see that several solutions may be discarded if they rendered infeasible by invoking the CP solver. While this may also happen with CP-Beam-ACO, the infeasible solutions are replaced by copies of a feasible solution. This is particularly effective since the performance of metaheuristics is dependent the number of solutions sampled and CP-Beam-ACO potentially generates many more solutions which can be used to bias the search.

This aspect also leads to a final advantage regarding feasibility. If feasible solutions are discontinued towards the end of the search tree, more feasible solutions are generated by CP-Beam-ACO. This improves the chances that a feasible solution will be found.

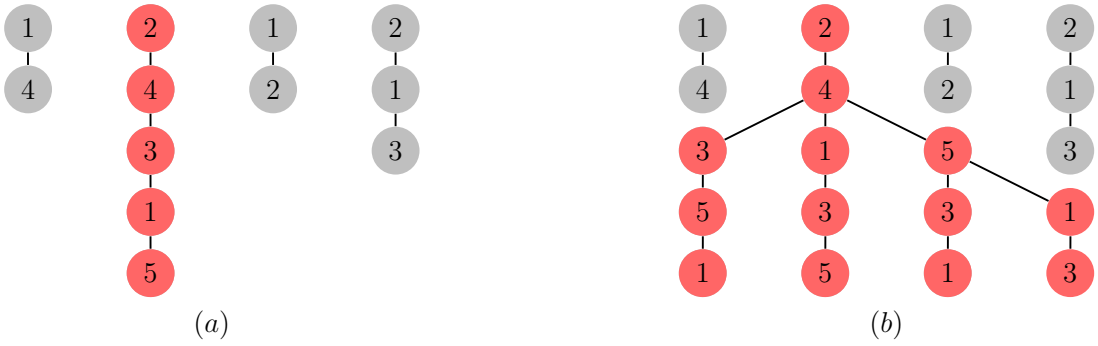


Figure 4.4: Here we see that CP-Beam-ACO potentially generates many more solutions. (a) shows that several solutions for CP-ACO may not lead to complete solutions. (b) where solutions may fail, e.g. after the first two components of the first solution, CP-Beam-ACO makes use of the feasible solutions to construct new solutions derived from the feasible ones.

We have discussed how CP-Beam-ACO may provide significant improvements to CP-ACO. However, a more complex hybrid introduces new aspects to consider when implementing the algorithm for a problem. For example, the estimate for the beam component must be carefully chosen. A good choice may achieve some or all of these improvements. These improvements may also be dependent on the problem characteristics.

A problem that we encounter is the same one that is carried over from the base algorithms, i.e., the choice of best parameters. Again, we point out here that there is no obvious way to determine the parameter settings but to examine some subset of the parameter space. For each study to be undertaken we specify how we determined the choices.

4.5 Conclusion

In this chapter we have motivated the need for an efficient version of the CP-ACO algorithm proposed by Meyer & Ernst (2004). Initially, hybridizations involving metaheuristics were discussed, where we see a clear advantage obtained by integrating with CP. Specifically, methods to deal with non-trivial hard constraints are often tedious to design and are problem specific. By devising a hybrid using CP we are able to deal with these hard constraints efficiently in a problem independent manner.

Although CP-ACO was shown to be effective, the algorithm requires large run times due to the propagation costs associated with CP. Within the current framework, the ACO component of the algorithm, often, repeatedly constructs the same solution which require the same propagation. Thus, by further hybridizing with Beam search the solution construction is parallelized, leading to CP-Beam-ACO where solutions within an iteration are never repeated.

CP-Beam-ACO proves advantageous in more than one respect. Firstly, the CP solver in only invoked for unique solutions. Secondly, a larger number of solutions are labeled compared to CP-ACO. Thirdly, more solutions are available to update the pheromones for the learning component and finally, dependent on the estimates of the beam component we may see a feasibility advantage, where more feasible solutions will occupy the beam near the end of the search tree. Having pointed out these advantages, it is worth noting that it is unlikely that all of these will be observed with every problem tested. In fact, for most problem types only one or two of these advantages may be seen. However, the advantage/s obtained can lead to significant improvements as we will demonstrate with case studies in the next chapter.

Applying CP-Beam-ACO is more complex than ACO or CP-ACO since we have further aspects of the algorithm to consider. This includes parameter settings which are hard to determine for the simpler algorithms to begin with. If these setting are carefully considered we may see significant improvements for CP-Beam-ACO. We consider three different problems (SMJS, MMJS and CS) in the next three chapters where CP-Beam-ACO is carefully applied to each of these problems and its performance is analysed.

Chapter 5

Case Studies

In order to compare CP-Beam-ACO to CP-ACO and the non-CP algorithms, we examine its performance on three different problems. These are single machine job scheduling with sequence dependent setup times (SMJS), resource constrained multiple machine job scheduling (MMJS) and car sequencing (CS). These problems consist of different characteristics and problem-specific characteristics cannot be expected to transfer from problem to problem. Each of these problems are either $\mathcal{NP}$-hard or $\mathcal{NP}$-complete and determining feasibility for SMJS and CS is know to be $\mathcal{NP}$-hard. We show here that CP-Beam-ACO based on stochastic sampling is the most effective algorithm for all these problems considering both optimality and feasibility.

5.1 Single Machine Job Scheduling

The single machine job scheduling (SMJS) problem with sequence-dependent setup times has received considerable interest within the AI and OR communities. It is closely related to other problems such as, the TSP with time windows. In fact, the problem is identical to the asymmetric TSP with time windows (Ascheuer, Fischetti and Grőtschel, 2001). Deciding if a feasible solution exists is known to be NP-complete (Savelsbergh, 1985) and finding an optimal solution is NP-hard (Ascheuer et al., 2001).

We evaluate the various implementations of ACO, CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS for the SMJS problem. The main result here is that CP-Beam-$\mathcal{MM}$AS significantly reduces the amount of computation time in comparison to CP-$\mathcal{MM}$AS. This is attributable the parallelization of the solution construction which allows storing intermediate solver states compactly. This allows partial solutions to efficiently transfer from one part of the search tree to another. This efficient use of the constraint solver leads to more computation time that can be devoted to the optimization/learning component.

5.1.1 Problem Definition

The problem is formally defined as follows. A single machine must process n jobs $J = \{1, ..., n\}$ with processing times $\overline{P} = \{\overline{p_1}, \ldots, \overline{p_n}\}$, release times $\overline{R} = \{\overline{r_1}, \ldots, \overline{r_n}\}$, and due times $\overline{D} = \{\overline{d_1}, \ldots, \overline{d_n}\}$.[1] The machine cannot process more than one job at the same time. Moreover, the processing of an job must not be interrupted. Furthermore, for each ordered pair (i, j) of jobs (where $i \neq j$) a setup time $\overline{st_{ij}}$ is given. Any permutation π of the n jobs represents a candidate solution to the problem. Given such a permutation π, where π_i denotes the job at position i, the so-called starting times $\hat{S} = \{\hat{s}_1, \ldots, \hat{s}_n\}$ and the ending times $\hat{E} = \{\hat{e}_1, \ldots, \hat{e}_n\}$ are well-defined. They can be determined recursively in the following way: $\hat{s}_{\pi_1} = 0$ and $\hat{s}_{\pi_i} = \max\{\hat{e}_{\pi_{i-1}} + \overline{st_{\pi_{i-1}\pi_i}}, \overline{r_{\pi_i}}\}$, where $\hat{e}_{\pi_{i-1}} = \hat{s}_{\pi_{i-1}} + \overline{p_{\pi_{i-1}}}$. The objective is to find a permutation π^* such that $f(\pi^*) = \hat{e}_{\pi^*_n}$ is minimal. Function $f()$ is commonly called the *makespan*.

5.1.2 $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System

The ACO variant used here is $\mathcal{MM}$AS. We also considered ACS and observed that $\mathcal{MM}$AS is better suited to the SMJS problem, particularly in terms of feasibility. In the context of this problem, variables are bound to jobs, a complete solution is a permutation/sequence of all jobs, and the pheromone trails are used to learn job successor relations. This is particularly meaningful for the ATSP since the aim is to minimize the cumulative setup times and learning successor relations are likely to result in high quality schedules. More specifically, the pheromone model $\mathcal{T}$ consists of a pheromone value τ_{ij} for each ordered pair (i, j) of jobs, where $i \neq j$. Additionally, pheromone model $\mathcal{T}$ consists of pheromone values τ_{0i}, $i = 1, \ldots, n$, where τ_{0i} represents the desirability of placing job i at the first position of the permutation under construction.

In addition to CP-ACS, $\mathcal{MM}$AS (Stűtzle and Hoos, 2000) is also implemented here in the hypercube framework (Blum and Dorigo, 2004). The reason for choosing $\mathcal{MM}$AS, in contrast to the choice of ACS in (Meyer and Ernst, 2004), is that $\mathcal{MM}$AS seems better suited than ACS for the hybridization with beam search (Blum, 2005). In the following we give a short technical description of this algorithm. For a more detailed introduction to the parameters and the functioning of this algorithm we refer to (Blum and Dorigo, 2004). At each iteration, $n_a = 10$ artificial ants construct (not necessarily feasible) solutions to the problem in form of permutations

[1] Bars indicate data for this study.

Algorithm 6 $\mathcal{MM}$AS for the SMJS problem

```
1: INPUT: A SMJS instance
2: π^bs := NULL(global best), π^rb := NULL(restart best)
3: // initialize the convergence factor and update flag
4: cf := 0, bs_update := FALSE
5: forall τ_ij ∈ T do τ_ij := 0.5 end forall
6: while termination conditions not satisfied do
7:    S_iter := ∅
8:    for j = 1 to n_a do
9:       π_j := ConstructPermutation()
10:      if π_j is a feasible solution then S_iter := S_iter ∪ {π_j} endif
11:   end for
12:   // Set the iteration best to the best solution
13:   π^ib := argmin{f(π)|π ∈ S_iter}
14:   if π^ib is a feasible solution then
15:      Update(π^ib,π^rb,π^bs)
16:      // update pheromone trails with the current best solutions
17:      ApplyPheromoneValueUpdate(cf,bs_update,T,π^ib,π^rb,π^bs)
18:      cf := ComputeConvergenceFactor(T)
19:      if cf ≥ 0.99 then
20:         if bs_update = TRUE then
21:            // reset the pheromones, restart best and update flag
22:            forall τ_ij ∈ T do τ_ij := 0.5 end forall
23:            π^rb := NULL, bs_update := FALSE
24:         else
25:            bs_update := TRUE
26:         end if
27:      end if
28:   end if
29: end while
30: OUTPUT: π^bs
```

of all jobs. The details of the algorithmic framework shown in Algorithm 6 are explained in the following.

ConstructPermutation(): A permutation π of all jobs is incrementally constructed from the first variable to the last. The solution construction stops when either the permutation is complete, or when it becomes clear that the resulting solution will be infeasible.[2] Given a partial permutation π with $i-1$ positions already filled, a job $k \in J \setminus \{\pi_1, \ldots, \pi_{i-1}\}$ for the i-th position in π is chosen as follows:

$$\mathbf{p}(\pi_i = k) = \frac{\tau_{\pi_{i-1}k} \times \eta_k}{\sum_{j \in J \setminus \{\pi_1,\ldots,\pi_{i-1}\}} \left(\tau_{\pi_{i-1}j} \times \eta_j\right)} \quad , \tag{5.1}$$

[2] If the CP solver renders the current partial solution as infeasible.

The obvious heuristics to use here would be the inverse of the due time (favours feasibility) or the inverse of the setup time (minimizes setup times). Therefore we make use of a two phase procedure where η_k is defined as the inverse of the due time of job k, i.e., $1/\overline{d_k}$, until a feasible solution is found. Once this happens η_k, respectively η_j, is exchanged by $\eta_{\pi_{i-1}k}$, respectively $\eta_{\pi_{i-1}j}$, and is defined as the inverse of the setup time between jobs π_{i-1} and k, respectively j. The first construction step, that is, when π is still empty, is a special one. In this case, the probability $\mathbf{p}(\pi_1 = k)$ is proportional to the pheromone value τ_{0k}. When a feasible solution is found there is no heuristic bias applied in the first selection step.

Update(π^{ib},π^{rb},π^{bs}): This procedure (as well as the pheromone update, see below) is only executed in case π^{ib} is a valid solution. It sets π^{rb} and π^{bs} to π^{ib} (i.e., the iteration-best solution), if $f(\pi^{ib}) < f(\pi^{rb})$ and $f(\pi^{ib}) < f(\pi^{bs})$.

ApplyPheromoneUpdate(cf,bs_update,$\mathcal{T}$,π^{ib},π^{rb},π^{bs}): Our algorithm potentially uses three different solutions for updating the pheromone values:[3] (i) the iteration-best solution π^{ib}, (ii) the restart-best solution π^{rb} and, (iii) the best-so-far solution π^{bs}. Their respective contribution to the update depends on the convergence factor cf, which provides an estimate about the state of convergence of the system. To perform the update, first an update value ξ_{ij} for each pheromone value $\tau_{ij} \in \mathcal{T}$ is computed: $\xi_e := \kappa_{ib} \cdot \delta(\pi^{ib}, i, j) + \kappa_{rb} \cdot \delta(\pi^{rb}, i, j) + \kappa_{bs} \cdot \delta(\pi^{bs}, i, j)$, where κ_{ib} is the weight of π^{ib}, κ_{rb} the weight of π^{rb}, and κ_{bs} the weight of π^{bs} such that $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1.0$. For $i, j = 1, \ldots, n$, the δ-function is the characteristic function, that is, $\delta(\pi, i, j) = 1$ if job j is scheduled directly after job i in permutation π, and $\delta(\pi, i, j) = 0$ otherwise. Similarly, when $i = 0$ and $j = 1, \ldots, n$, $\delta(\pi, 0, j) = 1$ in case j is the first job in permutation π, and $\delta(\pi, 0, j) = 0$ otherwise. Then, the following update rule is applied to all pheromone values:

$$\tau_{ij} = \min\left\{\max\{\tau_{\min}, \tau_{ij} + \rho \cdot (\xi_{ij} - \tau_{ij})\}, \tau_{\max}\right\} \quad ,$$

where $\rho \in (0, 1]$ is the learning rate, set to 0.1. The upper and lower bounds $\tau_{\max} = 0.999$ and $\tau_{\min} = 0.001$ keep the pheromone values always in the range $(\tau_{\min}, \tau_{\max})$, thus preventing the algorithm from converging to a solution. After tuning, the values for κ_{ib}, κ_{rb} and κ_{bs} are chosen as shown in Table 5.1.

[3] See (López-Ibáñez et al., 2009) which uses such an update scheme. In general, $\mathcal{MM}$AS update schemes make use of both the iteration and global best solutions, either independently in different phases or together (Dorigo and Stűtzle, 2004).

Table 5.1: The schedule used for values κ_{ib}, κ_{rb} and κ_{bs} depending on cf (the convergence factor) and the Boolean control variable *bs_update*.

	bs_update = FALSE				*bs_update* = TRUE
	$cf < 0.4$	$cf \in [0.4, 0.6)$	$cf \in [0.6, 0.8)$	$cf \geq 0.8$	
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1

Algorithm 7 Procedure ProbabilisticBeamSearch(θ, μ)

1: $B_0 = \{\pi = ()\}$
2: $t = 0$
3: **while** $t < n$ **and** $|B_t| > 0$ **do**
4: $\quad C =$ ProduceChildren(B_t)
5: $\quad$ **for** $k = 1, \ldots, \min\{\lfloor \mu \cdot \theta \rfloor, |C|\}$ **do**
6: $\quad\quad \pi^k =$ ChooseFrom(C)
7: $\quad\quad C = C \setminus \pi^k$
8: $\quad\quad B_{t+1} = B_{t+1} \cup \pi^k$
9: $\quad$ **end for**
10: $\quad B_{t+1} =$ Reduce(B_{t+1}, θ)
11: $\quad t \leftarrow t + 1$
12: **end while**
13: OUTPUT: **if** $t = n$ **then** $\operatorname{argmin}\{f(\pi) \mid \pi \in B_{n-1}\}$ **else** NULL

ComputeConvergenceFactor$(\mathcal{T})$: This function computes, at each iteration, the convergence factor (Blum, 2005)

$$cf = 2 \times \left(\left(\frac{\sum_{\tau_{ij} \in \mathcal{T}} \max(\tau_{\max} - \tau_{ij}, \tau_{ij} - \tau_{\min}))}{|\mathcal{T}| \times \tau_{\max} - \tau_{\min}} \right) - 0.5 \right) \tag{5.2}$$

The convergence factor cf can therefore only assume values between 0 and 1. The closer cf is to 1, the higher is the probability to produce the same solution over and over again.

5.1.3 Beam-$\mathcal{MM}$AS

In the following we explain how to obtain a Beam-ACO version from $\mathcal{MM}$AS of the previous section leading to Beam-$\mathcal{MM}$AS. Beam-$\mathcal{MM}$AS is obtained from Algorithm 6 by replacing lines 5-10 with $\pi^{ib} =$ ProbabilisticBeamSearch(θ, μ), which calls the probabilistic beam search procedure as shown in Algorithm 7. At the start of the procedure the beam only contains an empty permutation, that is $B_0 = \{\pi = ()\}$. At each iteration $t \geq 0$, the algorithm produces the set of all possible children C of the partial permutations that form part of the current beam B_t (see line 4,

ProduceChildren(B_t)). The partial solution is then extended by a choice for the next variable (π^k). Extending $\pi \in B_t$ means placing j at position $t+1$ of π. At each iteration, at most $\lfloor \mu \cdot \theta \rfloor$ candidate extensions are selected from C by means of the procedure ChooseFrom(C) to form the new beam B_{t+1}. At the end of each step, the new beam B_{t+1} is reduced by means of the procedure Reduce in case it contains more than θ partial solutions. The procedure stops when either B_t is empty, which means that no feasible extensions of the partial permutations in B_{t-1} could be found, or when $t = n-1$, which means that all permutations in B_{n-1} are complete. In the first case, the algorithm returns NULL, while in the second case the algorithm returns the solution with the best makespan.

Procedure ChooseFrom(C) uses the following probabilities for choosing a candidate extension $\pi_{t+1} = j$ from C:

$$\mathbf{p}(\pi_{t+1} = j) = \frac{\tau_{\pi_t j} \cdot \nu_{\pi_t j}}{\sum_{k \in J \setminus \{\pi_1, \ldots, \pi_t\}} (\tau_{\pi_t k} \times \nu_{\pi_t k})} \ , \tag{5.3}$$

The greedy function $\nu_{\pi_t j}$ assigns a heuristic value to each candidate extension π_t. In principle, we could use the earliest due date heuristic as in Eq. 5.1 for this purpose. However, when comparing two extensions $\pi_t = j \in C$ and $\pi'_t = k \in C$, their respective earliest due dates might be misleading in the case $\pi \neq \pi'$. As in (López-Ibáñez et al., 2009), we solved this problem by defining the greedy function $\nu(\cdot)$ as the inverse of the sum of the ranks of the earliest due date values that result from the complete construction of the partial solution corresponding to $\langle \pi, j \rangle$. For further details of this procedure see (López-Ibáñez et al., 2009).

Finally, the application of procedure Reduce(B_t) removes the worst $\max\{|B_t| - \theta, 0\}$ partial solutions from B_t. For this purpose one usually uses a lower bound for evaluating partial solutions. In fact, this lower bound is generally critical to the performance of the algorithm. However, in the case of the SMJS problem it is surprisingly difficult to find a lower bound that can be efficiently computed. In pilot experiments we discarded several obvious choices, including the minimum setup time estimate and the assignment problem relaxation (Carpaneto, Martello and Toth, 1988; Cormen et al., 2001). These are analysed in Chapter 6. Due to the lack of an efficient lower bound, we therefore use *stochastic sampling* for evaluating partial solutions. More specifically, for each partial solution a number N^{s} of complete solutions is sampled. This is done as follows. Starting from the respective partial solution, the solution construction mechanism of $\mathcal{MM}$AS is used to complete the partial solution in potentially N^{s} different ways. The only difference to the solution construction mechanism of $\mathcal{MM}$AS is that when encountering an infeasible partial solution the solution construction is not discarded. Instead, the partial solution is

Algorithm 8 Construct solutions using CP

```
1: i ← 0, feasible ← true
2: while i ≤ n & feasible do
3:    i ← i + 1
4:    repeat
5:       D = domain(π_i)
6:       j = select_job(D,τ)
7:       if i = 1 then
8:          feasible = post(π_i ← j ∧ ŝ_j ← 0)
9:       else
10:         feasible = post(π_i ← j ∧ ŝ_j ← max(r̄_j, ê_j + st̄_{π_{i-1},j}))
11:      end if
12:      if not(feasible) then post(π_i ≠ j)
13:   until D ≠ ∅ ∨ feasible
14: end while
15: if feasible then return π else return NULL
```

completed, even though it is infeasible. The value of the best one of the N^s samples is used as a measure of the goodness of the respective partial solution. Two measures are considered for comparing different samples: the number of constraint violations, and the makespan. The sample with the lowest number of constraint violations is considered best, and ties are broken with the makespan.

5.1.4 Integrating CP

Integrating CP into ACO for SMJS is shown in Algorithm 8. Extending the discussion seen in the previous chapter, the basic idea is to use constraint variables for the decision variables. We can then use a constraint solver in combination with a constraint model that captures the pertinent relationships between the decision variables to automatically keep track of the feasibility of (partial) solutions. We follow the study by (Meyer and Ernst, 2004) where CP-ACS was shown to be effective for the SMJS problem. The performance of the algorithm depends on finding feasible solutions and this is best achieved with a strong CP model which will guide the algorithm towards feasibility. This CP model suggested by Meyer & Ernst (2004) is discussed next.

The CP model for SMJS

The model maintains two sets of variables. The first set are those variables that were defined in the problem description in Section 5.1.1, that is, the positions π_i of permutation π, $\hat{S}$ and $\hat{E}$. A second set of variables from the CP side are introduced which define a stronger model that enhances propagation. Now for the permutation π, auxiliary variables corresponding to the constants are defined: $P = \{p_1, ..., p_n\}$,

release times $R = \{r_1, ..., r_n\}$, and due times $D = \{d_1, ..., d_n\}$. Note here that an index i of any of these variables refers to the job at position i in the permutation under construction, i.e., π_i. Additional variables for setup times $ST = \{st_2, ..., st_n\}$ are defined where the index refers to changeover times between consecutive jobs. For example, st_2 refers to the setup time from job π_1 to job π_2. Hence there are only $n-1$ such variables. Variables for the start times ($S = \{s_1...s_n\}$) and end times ($E = \{e_1...e_n\}$) corresponding to $\hat{S}$ and $\hat{E}$ are also defined. Now we can state the CP model. The jobs are first constrained to be unique:

$$\forall i: \quad \pi_i \in \{1, \ldots, n\} \wedge distinct(\pi_i)$$

Strong propagation is then achieved by combining these models by coupling the data and introducing constraints. The data are coupled with the following constraints:[4]

$$\begin{aligned}
&\forall i: && e_i = s_i + p_i \ \wedge \ s_i \geq r_i \ \wedge \ e_i \leq d_i \\
&\forall i > 1: && s_i \geq e_{i-1} + st_i \ \wedge \ st_i \in \{\overline{st_{jk}} \mid j, k \in \{1, \ldots, n\}\} \\
&\forall i: && d_i \in \{\overline{d_j} \mid j \in 1, \ldots, n\} \ \wedge \ r_i \in \{\overline{r_j} \mid j \in 1, \ldots, n\} \ \wedge \ p_i \in \{\overline{p_j} \mid j \in 1, \ldots, n\}
\end{aligned}$$

Once the id of the i^{th} job of the permutation is known the following data and variables can be bound:

$$\begin{aligned}
&\forall i > 1, k, l: && \pi_{i-1} = k \ \wedge \ \pi_i = l \Rightarrow st_i = \overline{st}_{kl} \\
&\forall i, k: && \pi_i = k \Rightarrow d_i = \overline{d_k} \ \wedge \ r_i = \overline{r_k} \ \wedge \ p_i = \overline{p_k} \ \wedge \ s_i = \min(\overline{r_k}, e_{i-1} + st_i) \\
&\forall i, k: && e_i > d_k \Rightarrow \pi_i \neq k
\end{aligned}$$

We make use of the high-level *cumulatives*$(\cdot)$ constraint, which a scheduling constraint in GECODE (Gecode, 2010). This constraint is initialized with all the data and interfaces with the model via $\hat{S}$ and $\hat{E}$. For cross propagation between the two models, the start, end and job id variables are coupled:

$$\begin{aligned}
&\forall i, k: && \pi_i = k \Rightarrow s_i = \hat{s}_k \ \wedge \ e_i = \hat{e}_k \\
&\forall i, k: && s_k > \hat{s}_i \vee \hat{s}_i < s_k \Rightarrow \pi_i \neq k
\end{aligned}$$

The integration of CP into Beam-$\mathcal{MM}$AS is very similar. In particular, CP is integrated into the phase of Beam-$\mathcal{MM}$AS which produces possible extensions to the current beam front (see Algorithm 9). The purpose is to restrict the construction of candidate solutions to those that are compatible with the problem constraints.

[4]Note that the bars on top labels imply constants and d_i is not the same as $\overline{d_i}$

Algorithm 9 ProduceChildren

1: INPUT: $(B_t, \mu, \mathcal{T})$
2: **for** $i \in B_t$ **do**
3: $k \leftarrow 0$, $D = \text{domain}(\pi^i_{t+1})$
4: **while** $k < \mu \wedge D \neq \emptyset$ **do**
5: $\overline{\pi} = \pi^i$
6: $j = \text{selectJob}(D,\tau)$
7: **if** $t = 0$ **then**
8: $feasible = \text{post}(\overline{\pi_{t+1}} \leftarrow j \wedge \hat{s}_j \leftarrow 0)$
9: **else**
10: $feasible = \text{post}(\overline{\pi_{t+1}} \leftarrow j \wedge \hat{s}_j \leftarrow max(\overline{r_j}, \hat{e_j} + \overline{st_{\pi_t,j}}))$
11: **end if**
12: **if** $(feasible)$ **then** $B_{t+1} = B_{t+1} \cup \overline{\pi}$
13: $k \leftarrow k + 1$, $D = D \setminus j$
14: **end while**
15: **end for**

The algorithm implemented here selects all feasible children and ranks them based on their due times as described earlier.

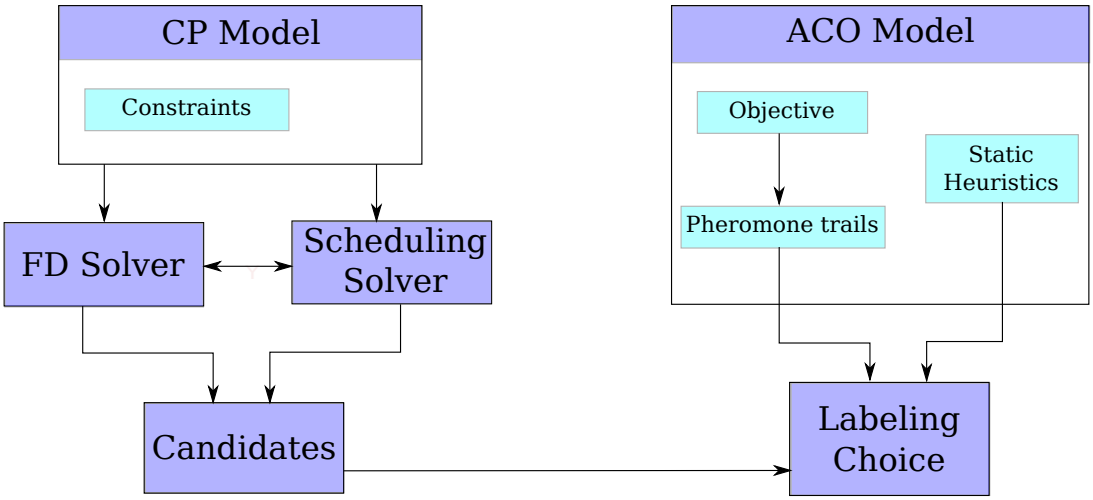


Figure 5.1: Here, we see coupling between the ACO and CP models. The CP solver builds a candidate list by invoking the finite domain and scheduling solvers. From the ACO side the pheromone trails and static heuristics are used and combined with the candidate list from the CP solver a choice is made.

Figure 5.1 shows the coupling between the two models to make a labeling choice. The CP solver invokes the finite domain and scheduling solvers and builds a candidate list. From the ACO side the pheromones and static heuristics are used to determine a labeling choice. This choice is approved only if it also exists in the candidate list held by the CP solver.

5.1.5 Experiments and Results

All experiments were run on the Monash Sun Grid and the Enterprisegrid[5] using the parameter sweep application and grid middleware broker Nimrod/G (Abramson, Giddy and Kotler, 2000). Each algorithm was applied for 30 times to each problem instance considered, allowing a running time of 10 hours for each application.

Problem Instances

The problem instances used in this paper include those used in (Meyer and Ernst, 2004) and additional ones from (Ascheuer et al., 2001). Individual files are labeled *xn.i* where x = label, n = instances size and i = index. For example, *W20.1* is the first file with *20* jobs. The first nine instances are real data taken from wine bottling in the Australian wine industry. There are no release times for this data and the instances are more constrained with decreasing index.

The second set of instances are for the ATSP with time windows and are taken from the literature (Ascheuer et al., 2001). The index refers to the number of time windows that have been removed. For example, the *RBG* problems are identical except that *RBG27.a.15* has time windows for *15* jobs. Hence, a larger index indicates a more tightly constrained instance.

The third set of instances are selected from the same source as above (Ascheuer et al., 2001) and aim to demonstrate the algorithms' consistency across a variety of instances. Due to time and resource constraints, instances with a maximum of 152 jobs were considered. Additionally, a number of smaller instances (< 27 jobs) which were not considered in (Meyer and Ernst, 2004) are not reported since the performance of all three algorithms is very similar, nearly always finding an optimal solution.

Parameter Settings

In the following we give a summary of the parameters settings that were chosen based on past experiments (Meyer and Ernst, 2004; Blum, 2005) and after tuning by hand. In CP-$\mathcal{MM}$AS the number of ants was set to 10 per iteration (based on (Meyer and Ernst, 2004)), the learning rate was set to $\rho = 0.01$ (tuned by hand). In $\mathcal{MM}$AS the upper bound for the pheromone levels was set to $\tau_{max} = 0.999$ and the lower bound to $\tau_{max} = 0.001$.

In CP-Beam-$\mathcal{MM}$AS we used $\theta = 10$, $\mu = 2.0$ and $N^s = 20$. The choice of $\theta = 10$ is motivated by the fact that CP-$\mathcal{MM}$AS uses this number of ants at

[5]The Nimrod project has been funded by the Australian Research Council and a number of Australian Government agencies, and was initially developed by the Distributed Systems Technology CRC.

each iteration. The setting of μ was adopted from past experiments with a similar problem (Blum, 2005). N^s was chosen based on initial experiments.

Results

The algorithms tested here include $\mathcal{MM}$AS, Beam-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS. Additionally, CP-ACS was also implemented and is briefly discussed. The major results of this chapter can be summarised as follows.

- CP-Beam-$\mathcal{MM}$AS is the best performing algorithm regarding solution quality and is superior to CP-$\mathcal{MM}$AS on 22 out of 43 instances. However, CP-$\mathcal{MM}$AS never fails on any run, whereas CP-Beam-$\mathcal{MM}$AS fails on a total of 4 out of 90 runs across 3 instances
- $\mathcal{MM}$AS performs poorly with respect to feasibility but is effective if feasibility is found
- Beam-$\mathcal{MM}$AS is slightly better than $\mathcal{MM}$AS when considering feasibility and is also effective if feasibility is found

Overall, CP-Beam-$\mathcal{MM}$AS is the best performing algorithm for the SMJS problem. It inherits the feasibility characteristics of CP-$\mathcal{MM}$AS and easily outperforms it in terms of optimality.

The results of the two algorithms are shown in Table 5.2, which has the following format. The first column contains the instance identifier. Then, for each algorithm we show four different columns. The first one of these four columns presents the best solution found across 30 runs (*best*), the second one shows the mean of the best solutions across all 30 runs (*mean*), the third one contains the associated standard deviations (*sd*), and the fourth one the number of times that the algorithm failed to find a feasible solution (*fail*) out of the 30 runs. In the three columns with heading *best* a value is marked in boldface in case no other algorithm found a better solution and at least one algorithm was not able to match this solution value. Similarly, in the three columns with heading *mean* a value is marked in boldface and italic if no other algorithm has a better mean and at least one algorithm has a worse mean.

In the following we delve into a comparison of both algorithms separated for the three instance sets. For the small instances ($<$ 25 jobs) both algorithms perform equally well except for instance W20.1 where CP-$\mathcal{MM}$AS performs poorly. In general, for the first set of instances, CP-Beam-$\mathcal{MM}$AS is the best performing algorithm on the medium-large instances. Even when outperformed, this algorithm always finds solutions close to the best. On the second set of instances the algorithms

perform equally well except for RBG27.a.3 where CP-$\mathcal{MM}$AS performs best and RBG27.a.15 where CP-Beam-$\mathcal{MM}$AS is the best by a large margin. On the third set of instances CP-Beam-$\mathcal{MM}$AS consistently outperforms the other two algorithms. Again, when the best or best average results are not obtained, CP-Beam-$\mathcal{MM}$AS always finds solutions whose costs are very close to the best. Furthermore, the algorithm is very robust, for many instances always finding the same best solution value across all 30 runs.

In Appendix D we also compare the original CP-ACS algorithm of Meyer & Ernst. First, comparing CP-ACS with the original version presented in (Meyer and Ernst, 2004) shows that our re-implementation performs competitively. Meyer & Ernst use labeling steps as the termination criteria, which results in the fact that the algorithm was run significantly longer than 10 hours for large problems. Despite the restricted time limit our re-implementation is competitive and is significantly worse than the original version only on problem instances W30.1 and RBG027.a.15. However, it is superior to the original version on several other problems (e.g., W30.2). This can be verified by comparing to the results provided in (Meyer and Ernst, 2004).

Concerning the comparison between CP-ACS and CP-$\mathcal{MM}$AS, the failure rate of CP-ACS is very high for instances with 48 or more jobs and this algorithm therefore does not scale well. In contrast, CP-$\mathcal{MM}$AS does not have such failure issues. In fact, CP-$\mathcal{MM}$AS is the only algorithm that never fails in finding a feasible solution. CP-ACS outperforms the CP-$\mathcal{MM}$AS (sometimes by a large margin) in terms of solution quality when feasible solutions are found. This may be explained by a certain amount of determinism in the solution construction of ACS.

Considering feasibility, as mentioned before, CP-ACS struggles on some of the largest instances, always failing on some (e.g. RBG050b) and nearly always failing on others (all instances of size greater than 67). While CP-$\mathcal{MM}$AS has no problems in this respect, CP-Beam-$\mathcal{MM}$AS fails in 4 out of 90 runs on three instances, demonstrating also a very low failure rate. Also see Figure 5.2 which shows the cumulative failures across the instances where all the algorithms have been tested. Clearly, the CP-based algorithms are superior with CP-ACS struggling on some instances.

The results for $\mathcal{MM}$AS and Beam-$\mathcal{MM}$AS with stochastic sampling on a subset of the instances are presented in Table 5.3. We only consider a subset of the instances here, including those from (Meyer and Ernst, 2004) and three additional instances, RBG050c, RBG055a and RBG067a. These results show that Beam-$\mathcal{MM}$AS outperforms $\mathcal{MM}$AS in terms of feasibility. Where feasibility is found $\mathcal{MM}$AS often finds higher quality solutions. A comparison with the CP-based algorithms shows that the non-cp algorithms struggle with feasibility. Also see Figure 5.2. However,

Table 5.2: Results of CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS for all considered instances. Statistically significant results ($p = 0.05$) are marked in boldface.

	CP-$\mathcal{MM}$AS				CP-Beam-$\mathcal{MM}$AS			
Instance	best	mean	sd	fail	best	mean	sd	fail
W8.1	8321	8321.00	0	0	8321	8321.00	0	0
W8.2	5818	5818.00	0	0	5818	5818.00	0	0
W8.3	4245	4245.00	0	0	4245	4245.00	0	0
W20.1	8914	9056.80	86.04	0	8504	**8504.00**	0	0
W20.2	5062	**5062.00**	0	0	5062	5068.00	11.33	0
W20.3	4312	**4312.00**	0	0	4312	4323.70	15.5	0
W30.1	8887	9068.50	127.67	0	8172	**8334.50**	65.52	0
W30.2	4682	4839.70	156.81	0	4527	**4666.30**	51.86	0
W30.3	4288	4364.20	48.36	0	4128	**4217.80**	35.46	0
RBG10.a	3840	3840.00	0	0	3840	3840.00	0	0
RBG16.a	2596	2596.00	0	0	2596	2596.00	0	0
RBG16.b	2094	2094.00	0	0	2094	2094.00	0	0
RBG21.9	4481	4481.00	0	0	4481	4481.00	0	0
RBG27.a.3	927	**927.10**	0.25	0	940	958.50	13.74	0
RBG27.a.15	1500	1543.60	18.54	0	1068	**1095.90**	19.28	0
RBG27.a.27	1076	1076.00	0	0	1076	1076.00	0	0
BRI17.a.3	1003	1003.00	0	0	1003	1003.00	0	0
BRI17.a.10	1031	1031.00	0	0	1031	1031.00	0	0
BRI17.a.17	1057	1057.00	0	0	1057	1057.00	0	0
RBG027a	5132	5177.70	18.51	0	5093	**5093.00**	0	0
RBG031a	3498	3498.00	0	0	3498	3498.00	0	0
RBG033a	3757	3757.00	0	0	3757	3757.00	0	0
RBG034a	3314	3362.00	31.1	0	3314	**3314.00**	0	0
RBG035a	3388	3446.10	44.41	0	3388	**3388.00**	0	0
RBG035a.2	3325	3325.00	0	0	3325	3325.00	0	0
RBG038a	5699	5914.90	83.45	0	5699	**5699.00**	0	0
RBG040a	5679	5680.70	5.74	0	5679	5679.00	0	0
RBG041a	3793	3906.30	89.41	0	3793	**3793.00**	0	0
RBG042a	3363	3491.20	71.08	0	3296	**3339.00**	20.6	0
RBG048a	9856	10019.30	89.26	0	9799	**9876.60**	44.8	1
RBG049a	13257	13401.10	72.09	0	13257	**13264.10**	12.9	0
RBG050a	12050	12050.90	5.11	0	12050	12050.00	0	0
RBG050b	12039	12155.40	69.65	0	12044	12126.60	33.6	0
RBG050c	11027	11115.10	65.7	0	10985	**11015.30**	22.6	1
RBG055a	6929	7045.50	64.32	0	6929	**6929.00**	0	0
RBG067a	10368	10485.00	65.51	0	10331	**10331.00**	0	0
RBG086a	16899	17028.80	85.06	0	16899	**16899.00**	0	0
RBG092a	12501	12530.10	35.77	0	12501	**12501.00**	0	0
RBG125a	14265	14383.20	98.45	0	14214	**14232.20**	25.2	0
RBG132	18524	18594.60	74.54	0	18524	**18524.00**	0	0
RBG132.2	18535	18764.50	96.63	0	18524	**18528.20**	14.1	0
RBG152	17455	17455.00	0	0	17455	17455.00	0	0
RBG152.2	17455	17505.80	64.3	0	17455	**17455.00**	0	2

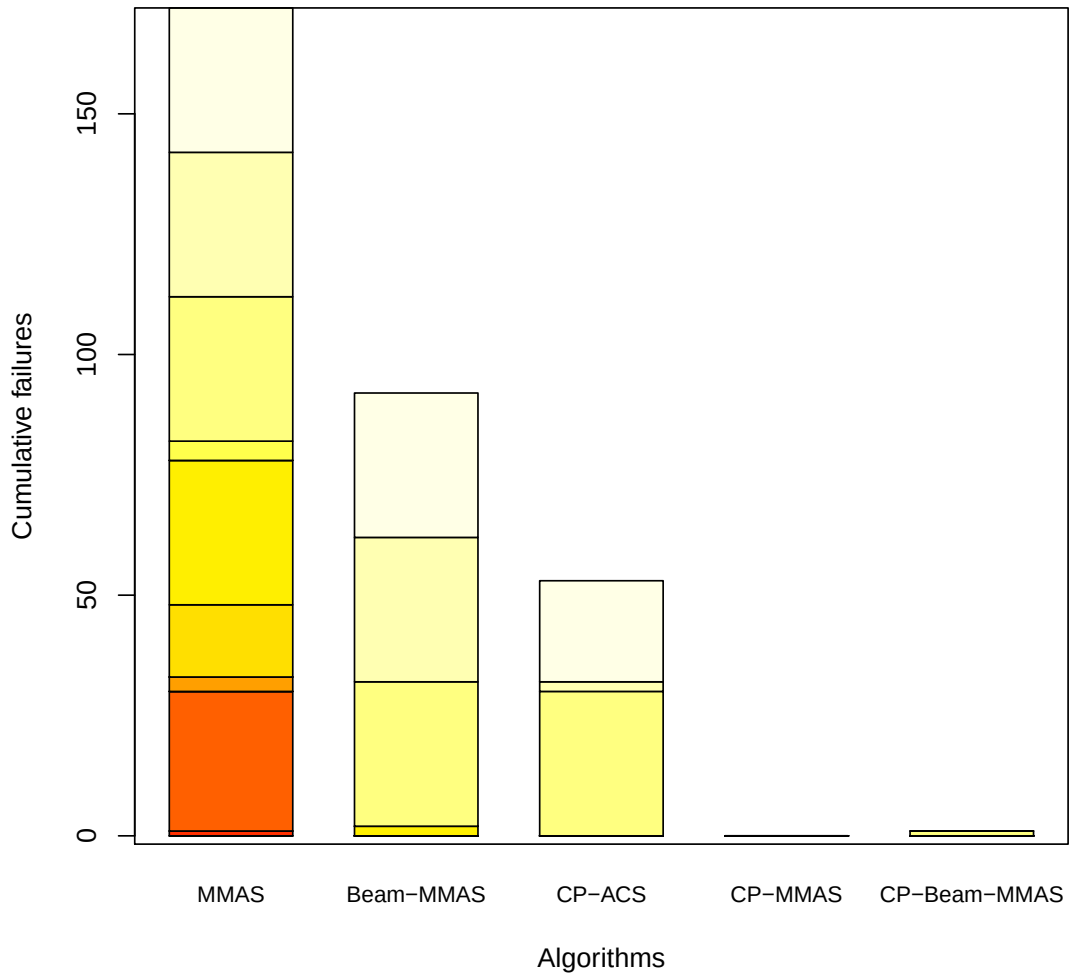


Figure 5.2: The cumulative failure results for $\mathcal{MM}$AS, Beam-$\mathcal{MM}$AS, CP-ACS, CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS are presented here; For each algorithm, each block corresponds to one problem instance where the shading/colour indicates the problem instance and the height of each block indicates the number of failures; The instances considered here are all of those from Meyer & Ernst (2004) and three additional instances, RBG050c, RBG055a and RBG067a.

when feasibility is found $\mathcal{MM}$AS is still very effective indicating that the learning component of ACO is valuable.

CP Propagation Costs

CP propagation costs are traded-off for stochastic sampling costs in CP-Beam-$\mathcal{MM}$AS. We therefore analyse the total cost for CP propagation versus the costs for stochastic sampling. We consider a number of instances of varying sizes from 8 to 67 jobs. For 100 iterations of the algorithm's execution we measure the time

Table 5.3: Results of $\mathcal{MM}$AS and Beam-$\mathcal{MM}$AS on a subset of the instances. Statistically significant results ($p = 0.05$) are marked in boldface.

	$\mathcal{MM}$AS				Beam-$\mathcal{MM}$AS			
Instance	best	mean	sd	fail	best	mean	sd	fail
W8.1	8321	8321.00	0.00	0	8321	8321.00	0.00	0
W8.2	5818	5818.00	0.00	0	5818	5818.00	0.00	0
W8.3	4245	4245.00	0.00	0	4245	4245.00	0.00	0
W20.1	8504	8518.83	1555.45	1	8504	8504.00	0.00	0
W20.2	5062	**5062.00**	0.00	0	5062	5078.00	14.04	0
W20.3	4312	**4312.00**	0.00	0	4312	4323.17	15.46	0
W30.1	8087	8087.00	1476.48	29	8012	8065.00	48.81	0
W30.2	4542	**4588**	20.94	0	4577	4654.50	57.76	0
W30.3	4183	4207.67	18.19	0	4163	4218.17	31.36	0
RBG10.a	3840	3840.00	0.00	0	3840	3840.00	0.00	0
RBG16.a	2596	2596.00	792.11	3	2596	2596.00	0.00	0
RBG16.b	2094	2094.00	0.00	0	2094	2094.00	0.00	0
RBG21.9	4481	4481.00	0.00	0	4481	4481.00	0.00	0
RBG27.a.3	927	**927.23**	0.94	0	944	968.17	14.99	0
RBG27.a.15	1068	1104.87	562.25	15	1131	1191.93	34.00	0
RBG27.a.27				30	1076	**1076**	272.99	2
BRI17.a.3	1003	1003.00	0.00	0	1003	1003.00	0.00	0
BRI17.a.10	1031	1031.00	0.00	0	1031	1031.00	0.00	0
BRI17.a.17	1057	1057.00	365.45	4	1057	1057.00	0.00	0
RBG050.c				30				30
RBG055.a				30				30
RBG067.a				30				30

required to for a CP call (i.e., calling the solver and obtaining its state) and the time required for stochastic sampling independently. The average results across the 100 runs are reported in Figure 5.3. As expected, given the complex CP model, the CP costs increase exponentially with problem size whereas stochastic sampling shows only a quadratic increase.

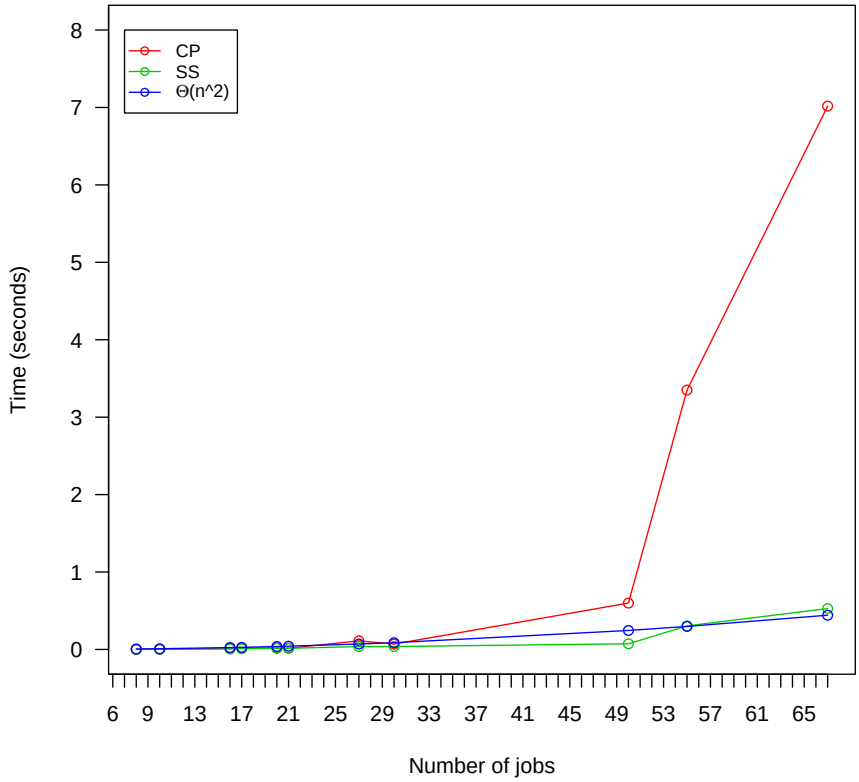


Figure 5.3: This figure shows the time needed (seconds) for CP (CP solver calls) versus SS (stochastic sampling). An $\Theta(n^2)$ trend-line is also shown. A randomly selected instance from each machine size was run for 10 iterations and the average results across the 10 runs are reported here. Stochastic sampling clearly requires a greater amount of time compared to the CP calls and this difference generally increases with machine size. Note that the results here do not require that feasible solutions are found.

5.1.6 Discussion

The results demonstrate overall that the best algorithm for the SMJS problem is CP-Beam-$\mathcal{MM}$AS. Out of 43 instances examined here Beam-ACO finds the best average solutions for 22 instances and is equal or approximately equal on 15 others. CP-$\mathcal{MM}$AS is more effective on three other instances.

For smaller instances (less than 20 jobs) all the algorithms perform equally well as expected. Considering instances of size greater than 20, CP-Beam-$\mathcal{MM}$AS performs better than or equally well to the other algorithms except for two instances. The reason for this improvement can be attributed to the beam search component of CP-Beam-$\mathcal{MM}$AS. First, the time spent in constraint propagation is reduced due to the parallel aspect of the solution construction in beam search. This leads to the fact that CP-Beam-$\mathcal{MM}$AS can spend more time in optimization in contrast to spending time in finding feasibility. Second, this parallel aspect of the solution construction in beam search also allows to discard partial solutions in favour of other partial solutions in different areas of the search tree that appear to be more promising.

Stochastic sampling was used as the preferred method of obtaining estimates. Further analysis of stochastic sampling is conducted in Chapter 6 and we briefly state the results here. We examined two different lower bounds in place of stochastic sampling which do not prove effective with respect to feasibility or solution quality. Furthermore, we see that the samples generated with uniform sampling are very effective but still slightly worse than using pheromones. This is an unexpected result but shows that samples which are generated without learning can still be useful.

In summary, CP-Beam-$\mathcal{MM}$AS is able to make an efficient trade-off between the use of CP for feasibility and the stochastic-heuristic component for finding high quality solutions. Since the CP overhead is large, the CP-ACO variants spend more time propagating constraints compared to CP-Beam-$\mathcal{MM}$AS and are thus less efficient.

5.1.7 Conclusion

We have shown in this study that a hybrid algorithm resulting from the combination of Beam-ACO with constraint programming is an effective algorithm for the SMJS problem providing excellent results optimizing makespan. This method uses CP effectively parallelizing the ACO solution construction and exploiting dependencies between partial solutions to find high quality solutions. In particular, we see that CP-Beam-$\mathcal{MM}$AS outperforms CP-$\mathcal{MM}$AS maintaining its feasibility advantage given the same time-frame. These algorithms easily outperform the non-CP algorithms since $\mathcal{MM}$AS or Beam-$\mathcal{MM}$AS struggle with feasibility for relatively large problems.

Stochastic sampling was used to obtain estimates for the beam component and we see that this relatively simple procedure is effective. In order to gain better understanding of stochastic sampling we analyse this method for the SMJS problem

in Chapter 6. Furthermore, we also analyse bounds known for the SMJS problem in this chapter to determine whether or not they provide effective estimates.

5.2 Resource Constrained Job Scheduling

Resource constrained job scheduling has been widely researched in the last 30-40 years with one of the first instances being Johnson's PhD thesis (Johnson, 1967). The focus of this research has mainly been to minimize makespan as the objective (e.g. the same objective as the SMJS problem). However, recently minimizing tardiness has also been given some attention (Ballestin and Trautmann, 2008). To get an idea of the variants of similar problems please refer to (Brucker, Drexl, Mohring, Neumann and Pesch, 1999; Demeulemeester and Herroelen, 2002; Schwindt, Neumann and Zimmermann, 2003).

The variant of the problem considered here is resource constrained multiple machine job scheduling which requires scheduling a number of jobs on multiple machines. The machines may only process one job at a time but, additionally, each job requires resources that may be commonly used by all the jobs during the current time point. Therefore, at any single time point the cumulative resource requirements of the executing jobs must not exceed the available resource. This sort of resource has been referred to as a renewable resource (Brucker et al., 1999) and this particular problem was initially motivated from the following scenario. Consider several mining sites which are not connected to the main electricity grid. All these sites may require power from a single power plant which provides electricity (the shared resource) to the operations at the mines (machines). The power plant may only be able to provide a limited amount of electricity (e.g., the government may impose a cap for environmental reasons) and the operations must therefore be scheduled in such a way that they do not exceed the available power. This may result in some jobs being delayed beyond their due time and hence the problem may be formulated as one where we aim to minimize tardiness. However, some operations might have to complete by a certain time and we incorporate these restrictions as additional hard problem constraints.

In this chapter we examine methods based on ACO, CP and beam search to determine if they are effective when minimizing tardiness. In order to do this we determine an ACO model based on past studies and suggest a CP model via the use of the high-level cumulatives scheduling constraint. In order to examine the beam search variants we test various bounds (known from literature) including stochastic sampling which is implemented in a similar manner to how it was implemented for the SMJS problem. We also re-implement the simulated annealing algorithm from (Singh and Ernst, 2011) to be able to make a straight-forward comparison with the resulting algorithm.

5.2.1 Problem Definition

The multiple machine job scheduling (MMJS) problem can be formally defined as follows. A number of machines $\mathcal{M} = \{m_1, \ldots, m_l\}$ must process a number of jobs $\mathcal{J} = \{j_1, \ldots, j_n\}$. Each job i is associated with the following data:

- r_i: release time of the job
- p_i: processing time of the job, i.e., the number of time units the job will spend on its machine
- d_i: due time of the job which is the desired time but not enforced as a hard constraint
- w_i: weight of the job (tardiness penalty for completing after d_i)
- g_i: resource used by the job when executing on a machine
- m_i: machine that the job must be scheduled on

Each job may only be scheduled after its release time and it must only be executed on its machine. Each machine is capable of executing only one job at a time. There is no setup time between jobs and once scheduled, no job may be stopped before it completes (i.e., no preemption). In addition to the resource constraints, precedence constraints between jobs may exist. The set of precedences is denoted by PR. For two jobs $i, j \in \mathcal{J}$ if i precedes j ($i \rightarrow j$), then j may only start after i completes. Except for w_i all data are integral.

A sequence of all jobs is specified by a permutation of the jobs π. Let $\sigma(\cdot)$ be a mapping from sequences to schedules. A feasible schedule of π, $\sigma(\pi)$, assigns start times ($\mathcal{S} = \{s_1, \ldots, s_n\}$) and end times ($\mathcal{C} = \{c_1, \ldots, c_n\}$) to all the jobs such that $s_i \geq r_i$ and $s_j \geq s_i + p_i = c_i$ if $i \rightarrow j$. Let P_t be the set of jobs either starting at time t or being processed at time t:

$$P_t = \{j | s_j \leq t < s_j + p_j, j \in \mathcal{J}\} \tag{5.4}$$

$\sigma(\pi)$ is considered resource feasible if

$$\forall t \sum_{k \in P_t} g_k \leq \mathcal{G} \tag{5.5}$$

where g_k is the amount of resource required by the job scheduled at time point t and $\mathcal{G}$ is the maximum amount of resource available across all the machines. This equation specifies that resources requirements of all jobs executing at the same time

must not exceed the available resource. The objective is to minimize the total weighted-tardiness of a resource feasible schedule $\sigma(\pi)$

$$T(\sigma(\pi)) = \sum_{i=0}^{n} w_{\pi_i} \times T(\sigma(\pi_i)) \tag{5.6}$$

where $T(\sigma(\pi_i))$ is the tardiness of the job π_i, $max(c_{\pi_i} - d_{\pi_i}, 0)$.

In addition to the above specification we include hard constraint in the form of hard due times. The motivation for these additional constraints is that several or all of the jobs may be required to complete within a specific time. The original due times still remain as soft constraints and can be interpreted as the time by which we would like to complete the job (allowing a penalty for surpassing this time) whereas the new constraints specify that jobs have to be finished by this time. More formally, we define $\bar{d}_i, i \in \{1, \ldots, \mathcal{J}\}$ and require that $s_i + p_i \leq \bar{d}_i$. These hard due times are an addition to the problem tackled by (Singh and Ernst, 2011).

5.2.2 Ant Colony System

In the context of the problem at hand, we specify a model suggested by (den Besten, Stűtzle and Dorigo, 2000). This study examines an ACO algorithm for the single machine problem with the total weight tardiness objective. Hence, the ACO model in this study is a plausible model to use here.[6] The pheromones $\mathcal{T}$ consist of pheromone values τ_{ij} for each job j and variable i. Algorithm 10 shows the ACS implementation for MMJS.

We chose to implement ant colony system (ACS). However, we also tested $\mathcal{MM}$AS to determine its effectiveness on this problem but ACS was more effective on this problem based on initial results. Algorithm 10 is similar to the previous versions of ACS that we have seen so far. Here, we briefly describe the procedures used by this algorithm and highlight some of the diferences to the usual ACS. The parameter settings will be discussed in Section 5.2.7.

ConstructPermutation(): The permutation π of the jobs is constructed by selecting a job for each variable. A solution is considered complete when all the variables have unique jobs assigned to them. The selection of a job proceeds as follows. We generate a random number $q \in (0, 1]$ and compare this with a pre-defined parameter q_0. For the i^{th} variable, π_i, if $q > q_0$, a job k for variable i in π is greedily selected according to

$$k = \max_{j \in \mathcal{J} \setminus \{\pi_1, \ldots, \pi_{i-1}\}} \tau_{ij} \times \eta_j \tag{5.7}$$

[6]Note that we also briefly tested other learning models which shown not to be as effective.

Algorithm 10 ACS for the MMJS problem

```
1: INPUT: A MMJS instance
2: π^bs := NULL (global best)
3: initialize 𝒯
4: while termination conditions not satisfied do
5:    S_iter := ∅
6:    for j = 1 to n_a do
7:       π_j := ConstructPermutation()
8:       CompletePlacement(π_j)
9:       S_iter := S_iter ∪ {π_j}
10:   end for
11:   // Set the iteration best to the best solution
12:   π^ib := argmin{f(π)|π ∈ S_iter}
13:   π^ib := LocalSearch()
14:   Update(π^ib,π^bs)
15:   // Update the pheromone trails with the current best solution
16:   PheromoneUpdate(𝒯,π^bs)
17:   // Determine if the pheromones have converged
18:   cf := ComputeConvergence(π^ib)
19:   // reset the pheromones if they have converged
20:   if cf = true then initialize 𝒯 end if
21: end while
22: OUTPUT: π^bs
```

otherwise, select k ($k \in \mathcal{J} \setminus \{\pi_1, \ldots, \pi_{i-1}\}$) from the following distribution

$$\mathbf{p}(\pi_i = k) = \frac{\tau_{ik} \times \eta_k}{\sum_{j \in \mathcal{J} \setminus \{\pi_1, \ldots, \pi_{i-1}\}} (\tau_{ij} \times \eta_j)} \tag{5.8}$$

η_k is defined as w_k/d_k. This biases the selection of jobs with large weights and earlier due times. While there are other plausible heuristics such as w_k/p_k, initial tests showed that the above heuristic was the most effective for this problem. Note that the selection of a job does not take precedences into account and these details will be discussed later.

Every selection of a job j to a variable i has an associated update to the pheromones:

$$\tau_{ij} = \tau_{ij} \times \rho + \tau_{min} \tag{5.9}$$

where ρ is a learning rate that is chosen so that the pheromones reduce gradually. This form of update essentially allows the algorithm to diversify to avoid geting stuck in a local optimal. $\tau_{min} = 0.001$ is a small value that ensures that no pheromone value is too small such that it will not be considered in future solution constructions.

CompletePlacement(): Once a sequence for the current solution has been specified the schedule $\sigma(\pi)$ is determined. This is done using a placement scheme (see Section 5.2.3) which satisfies precedence and resource constraints to generate a resource feasible schedule.

LocalSearch(): Incorporating local search within the algorithm was motivated by the fact that the SA algorithm described previously is very effective on this problem. This algorithm essentially interleaves two neighbourhood moves in various ways. Over here, we consider these moves but apply them sequentially a constant number of times. The local search is applied to the best solution found in an iteration. The two neighbourhood moves are as follows:

- Random swapping: this is the most basic move and works as follows. Two values $i, j \in 1, \ldots, |\mathcal{J}|$ are selected uniformly randomly the corresponding jobs π_i and π_j are swapped creating a new solution - $\hat{\pi}$. The old solution is replaced with the new one if the the new solution has improved tardiness, $f(\sigma(\hat{\pi})) < f(\sigma(\pi)) \Rightarrow \pi = \hat{\pi}$. This move is applied a constant number of times.

- β-sampling: given the best solution from the above procedure, β-sampling (Valls, Quintanilla and Ballestin, 2003) is applied to it a constant number of times. The basic idea is to select a sub-sequence of the permutation starting at index i (selected uniformly from all the indexes) and move these jobs to the end of the permutation. All subsequent jobs are moved up to i. Figure 5.4 demonstrates β-sampling on a permutation of 10 jobs with sample size 4.

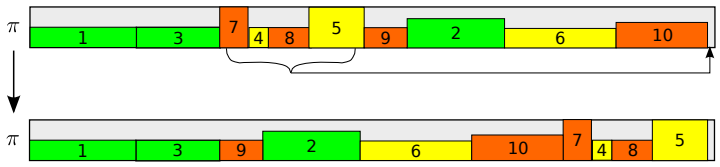


Figure 5.4: This is demonstration of β-sampling for a permutation of 10 jobs. Using a sample size of 4 and the starting index $i = 2$, jobs 7,4,8 and 5 move to the end of the permutation. They maintain their relative sequence as they are shifted. Jobs of the same colour are required to be placed on the same machine.

Update(π^{ib},π^{bs}): This procedure sets π^{bs} to π^{ib} if $f(\sigma(\pi^{ib})) < f(\sigma(\pi^{bs}))$ where $f(\sigma(\pi^{ib}))$ is the cost of the iteration best solution (i.e., the weighted tardiness). For the algorithms that do not use CP we first minimize over the number of constraint violations ($\nu(\sigma(\pi^{ib})) < \nu(\sigma(\pi^{bs}))$) determined from a sequence followed by weighted tardiness.

ApplyPheromoneUpdate($\mathcal{T}$,π^{bs}): The algorithm updates the pheromones based on the global best solution. All components (i, j) appearing in the global best solution are used to update the corresponding components in the pheromone matrix:

$$\tau_{ij} = \tau_{ij} \times \rho + \delta_{\pi^{bs}} \tag{5.10}$$

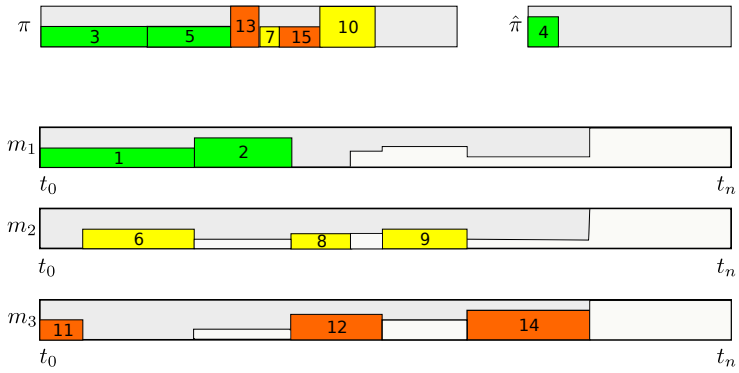
where $\delta_{\pi^{bs}} = Q/f(\pi^{bs})$ and Q is determined such that $0.01 \leq \delta_{\pi^{bs}} \leq 0.1$. This is the reward factor and ensures that for all instances the reward factor is of the same order. ρ is the learning rate as specified earlier and is defined to be 0.1 for this study. The pheromones in the CP based algorithms are not rewarded while a feasible solution is not found but is a global best solution is found it is always rewarded even if the current iteration finds no feasible solution.

ComputeConvergence(π^{ib}): Here, we compute a convergence measure to determine if the construction mechanism is repeatedly producing the same solution. For this purpose, a list of the past m solutions, $l_{\pi^{ib}}$ is maintained. The current iteration best π^{ib} is compared to these solutions and if they all have the same cost the pheromones are re-initialized: $f(\pi^{ib}) = f(k), k \in l_{\pi^{ib}} \Rightarrow \tau_{ij} = 0.5 \ \forall i, j$.

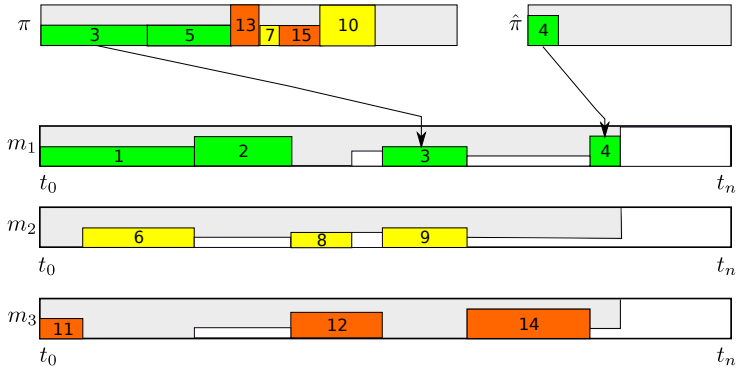
5.2.3 Placement Schemes

(Singh and Ernst, 2011) describe the serial and parallel palcement schemes for a sequence of jobs. They showed that the serial scheme is usually superior and we therefore focus on this scheme here. The serial method places jobs on their machines starting at time point t_0. Each job is tested for placement at a free point on the time-line. If it satisfies the resource and precedence constraints and a sufficiently large sequence of time points are free, the job is immediately placed. See Figure 5.5. Here, an example is presented for the three machine fifteen job problem. Figure 5.5 (a) shows the current state of the placement where a number of jobs are still left in the permutation (π) to schedule. The waiting list ($\hat{\pi}$) has one job at this point. This list consist of those jobs which arrived early in the sequence, but due to precedence constraints must be scheduled after jobs that are still in the sequence. All such jobs are placed in the list. After any job from the sequence is scheduled the list is examined to determine if any job may be placed. These waiting jobs are placed as soon as possible. This is observed in Figure 5.5 (b) where job 3 is placed on machine one (m_1) and job 4 from the waiting list is placed at the earliest possible time after job 3. In this example, job 4 is placed much later than job 3 and after job 14 since job 14 is uses most of the available resource which job 4 is unable to share.

The parallel placement scheme works as follows. Starting at t_0 the sequence of jobs is examined to determine if any job can be placed. Once a job is placed the



(a)



(b)

Figure 5.5: This is an example of a three machine 15 job problem. (a) Jobs $1-5$ must be placed on machine 1 (m_1, green), $6-10$ on machine 2 (m_2, yellow) and $6-10$ on machine 3 (m_3, orange). π is the sequence of jobs remaining to be scheduled and $\hat{\pi}$ is the waiting list. The height of the machines reflect the total resource available at each time point and the sum of the heights of the jobs scheduled across a time point must be less than or equal to the height of any one of the machines. The region in white shows the available resource in any machine. t_0 is the point at which the scheduling starts and t_n is the last point on the time-line. In this example job 3 must be placed before job 4. (b) Once job 3 is placed, job 4 is placed immediately in the first available location. Note that job 4 has to wait for job 14 to complete since the resource requirement for both these jobs to execute in parallel is greater than the total available resource.

sequence is examined again until no possible job can be placed on any machine (due to the resource constraint) at this time point. The procedure then moves along to the next time point and attempts to place jobs in the same fashion. This procedure continues until the job list is empty.

A consequence of this placement scheme is that the jobs need not be sequenced in order of precedence by the ants. While the precedences could be maintained when constructing sequences, it was found in initial tests that the solution quality found with this procedure was worse than when the precedences were allowed to

be violated and handled at the placement phase. This is despite both procedures permitting the optimal sequence, although, the first procedure allows a much larger number of sequences that map to the optimal solution.

5.2.4 Simulated Annealing

For the sake of completeness, we provide the SA algorithm suggested by (Singh and Ernst, 2011) (see Algorithm 11). The algorithm is briefly described here and we refer the author to the original paper for the complete details.

We obtain an initial temperature t, the procedure for which is described in (Singh and Ernst, 2011). The algorithm now repeats until the time limit is reached (line 3). Between lines 8-20 the algorithm randomly swaps jobs and accepts the new solution if it is an improvement on the original solution. However, if the new solution quality is worse, it is still accepted with some probability (line 15). Outside this loop the $\beta-$Sampling procedure is applied at line 22. These steps are repeated five times (lines 5-24) after which the best solution is mutated (line 25).

Note that the solution quality is determined by two factors. The first is the number of violations of the hard constraints of a sequence and the second is the weighted tardiness. A new solution is considered an improvement if it has a smaller violation count or an equal violation count with smaller weighted tardiness.

Mutate(π^{bs}): This procedure, with equal probability, perturbs π^{bs} in the following way:

- apply $\beta-$Sampling() several times and return best solution found
- uniformly randomly swap any two jobs n_{swaps} times[7] and return the best solution found
- return a new random list

5.2.5 The CP model

The CP model devised here uses two high level constraints in addition to a number of low level constraints. Firstly, both high level constraints are an extension of *cumulatives*$(\cdot)$. Specifically, we have for each machine $m \in \{1, \ldots, \mathcal{M}\}$, *cumulatives*$(s^m, p^m, c^m)$ which specifies that all jobs $i, j \in \{1, \ldots, \mathcal{J}\}$ in machine m must have $s_j^m \geq s_i^m + p_i^m = c_i^m$ or $s_i^m \geq s_j^m + p_j^m = c_j^m$. This constraint does not specify any relationship with the resource constraint and therefore *cumulatives*(s, p, c, g) is also used. The execution times are allowed to overlap if the cumulative resource

[7] n_{swaps} is randomly chosen from $\{1, \ldots, n\}$

Algorithm 11 Simulated Annealing the MMJS problem

INPUT: A MMJS instance
initialize t_0
while termination conditions not satisfied **do**
  $k = 0, t = t_0$
  while $k < 5$ **do**
    tardiness $= \infty$
    $\pi^{rb} = \pi^{ib}$
    while $i < 5000$ **do**
      swap π^{rb}_l and π^{rb}_m $(l, m \in \{1, \ldots, |\mathcal{J}|\})$
      if $f(\sigma(\pi^{rb})) <$ tardiness **then**
        tardiness $= f(\sigma(\pi^{rb}))$
        $\pi^{ib} = \pi^{rb}$, $i = 0$
      else
        $\Delta = f(\sigma(\pi^{rb})) - tardiness$
        if $e^{-\Delta/t} < random()$ **then**
          swap π^{rb}_l and π^{rb}_m
        end if
        $i = i + 1$, $t = t \times 0.95$
      end if
    end while
    if $f(\sigma(\pi^{ib})) < f(\sigma(\pi^{bs}))$ **then** $\pi^{bs} = \pi^{ib}$ **end if**
    $\pi^{ib} = \pi^{bs} \rightarrow \beta-$Sampling();
    $k = k + 1$
  end while
  $\pi^{ib} = \mathsf{Mutate}(\pi^{bs})$
end while
OUTPUT: π^{bs}

used during a particular time is bounded by the total resource available. Combining the above constraints enforces all the start times on a single machine do not overlap and also that the cumulative resources used at a particular time point are always less than the available resource.

Additionally, we specify the following constraints:

$$\forall m \in \{1, \ldots, \mathcal{M}\} \wedge \forall i \in \{1, \ldots, \mathcal{J}^m\} : \quad s_i \geq r_i \wedge s_i + p_i = c_i \wedge c_i \leq \hat{d}_i$$
$$\forall m \in \{1, \ldots, \mathcal{M}\} \wedge \forall i, j \in \{1, \ldots, \mathcal{P}^m\} : \quad i \rightarrow j \Rightarrow s_j \geq s_i + p_i$$

The first set of constraints specify the relationships between the start and end times of the jobs on a machine m. In particular, the job must start after or at its release time, its completion time is the sum of its start time and processing time and its completion time has to be less than the hard due date. The second set of constraints specify the precedence between jobs on a machine.

Algorithm 12 Construct solutions using CP

1: $i \leftarrow 0$, *feasible* $\leftarrow$ *true*
2: **while** $i \leq n$ & feasible **do**
3: $i \leftarrow i + 1$
4: $D = \text{domain}(\pi_i)$
5: **repeat**
6: $j = \mathsf{selectJob}(D,\tau)$
7: **if** PR_j in $\sigma(\pi)$ **then**
8: *feasible* = updateJobs(π_i, j, $\hat{\pi}$)
9: **else**
10: *feasible* = *true*, *append* j *to* $\hat{\pi}$
11: **end if**
12: **if** not(*feasible*) **then** post($\pi_i \neq j$)
13: $D = D \setminus j$
14: **until** $D \neq \emptyset \vee$ *feasible*
15: **end while**
16: Return π

5.2.6 Integrating Constraint Programming

The integration here is along similar lines to that of the previous CP integrations seen so far (see Algorithm 12). The high-level algorithm is essentially the same as Algorithm 4 from Chapter 2. However, a difference appears here during the solution construction phase compared to the algorithm for the SMJS problem.

We first recall how this was done in the SMJS problem (Meyer and Ernst, 2004). For each variable in the sequence, we select a job by sampling the pheromones. This selection is posted to the CP solver and we continue to select jobs if the solver state is not in failure. If a selection is inconsistent with the solver this job is discarded from the current candidate list and a new selection is made. If we are able to bind all the variables to values we produce a feasible solution.

The difference here is that we keep a waiting list of jobs ($\hat{\pi}$ in Algorithm 12). These are jobs that may only be scheduled if their preceding jobs have been scheduled. Therefore, a selection of a job does not automatically invoke the solver. This is only the case if the job is scheduled when selected. On line 7, PR_j is the set of preceding jobs of j and j may only be scheduled if its preceding list of jobs have been scheduled. If j can be scheduled, this may allow a number of other jobs to be scheduled (updateJobs($\cdot$)). That is, there may be jobs on the waiting list which are free to be scheduled once j has been placed. The placements for all jobs that are scheduled are posted to the solver, immediately, to determine feasibility.

The result of this scheme is that a number of jobs may be sequenced without being scheduled. Since the solver is only invoked when they are placed, it may happen that the new job being placed may already be infeasible as a result of the waiting jobs. Hence, this solver potentially leaves it too late to allow jobs to be

Algorithm 13 Probabilistic Beam Search with CP

```
1: INPUT: (θ, μ, 𝒯)
2: // Initialize the beam with θ empty solutions
3: B_0 = {π_1 = (), ..., π_θ = ()}
4: t = 0
5: while t < n and |B_t| > 0 do
6:    for i ∈ B_t do
7:       k ← 0, D = domain(π^i_{t+1})
8:       while k < μ ∧ D ≠ ∅ do
9:          (π̄, π̂) ← π^i, feasible = true
10:         j = selectJob(D, 𝒯)
11:         // If the predecessors of j have been scheduled
12:         if PR_j in σ(π̄) then
13:            feasible = updateJobs(π̄, π̂, j)
14:         else
15:            π̂ ← append(π̂, j)
16:         end if
17:         if feasible then B_{t+1} = B_{t+1} ∪ (π̄, π̂)
18:         k ← k + 1, D = D \ j
19:      end while
20:   end for
21:   B_{t+1} = Reduce(B_{t+1}, θ)
22:   t ← t + 1
23: end while
24: OUTPUT: argmax{f(π) | π ∈ B_{n−1}}
```

sequenced. However, given that improved sequences are obtained by not maintaining precedences while sequencing jobs, there is no possible way to post to the solver immediately. In terms of obtaining feasibility, the current scheme still provides an effective way to tackle the problem by immediately invoking the solver for most of the jobs.

CP-Beam-ACS

CP-Beam-ACS is implemented in a similar fashion to the implementation for SMJS. For the next variable in the partial solution, a number of children are selected based on the pheromone information biased by a heuristic factor. The total set of solutions now includes $\theta \times \mu$ candidates. From these the best θ solutions are selected in a greedy fashion using ϕ. Note that ϕ is also computed for a job that is sent to the waiting list. This set is further reduced to θ using the estimate ϕ.

See Algorithm 13. This algorithm is different in several aspects compared to the previous probabilistic Beam search algorithms we have seen so far. Now we have a solution represented by two sequences $(\overline{\pi}, \hat{\pi})$ where the first is the current sequence of jobs and the second sequence is the waiting list of jobs. A job is selected in line 9

and this selection is posted to the solver only if its preceding jobs (PR_j) have been scheduled. This is done in line 11, updateJobs($\bar{\pi}$, $\hat{\pi}$, j). Here, the sequence $\bar{\pi}$, the waiting list $\hat{\pi}$ and selected job j are passed as arguments. Once j is scheduled, all those waiting jobs that are now free to be placed are immediately scheduled. If a job is not scheduled it is placed in the waiting list (line 13). The new sequence is placed in the beam if it is either feasible or if the new selection is waiting (line 15). The algorithm then proceeds as usual by using estimates at line 19 to eliminate non-promising solutions.

With the SMJS problem, biasing the search more strongly with the initial heuristic selection was advantageous. This method, however, also suffered from the inability of the algorithm to examine certain solutions due to this scheme, potentially ruling out optimal areas of the search space. Therefore, to avoid this problem the greedy heuristic selection step is discarded. Furthermore, the three stage selection was tested with this problem, unsuccessfully, suggesting that the heuristics tested for this problem are not as effective here as the heuristics were for the SMJS problem.

5.2.7 Experimental Setting

The results for experiments with ACS, Beam-ACS, CP-ACS, CP-$\mathcal{MM}$AS and CP-Beam-ACS are reported here. We also compare these results to the original SA algorithm. In order to select an appropriate ACO variant we first conducted experiments with CP-ACS and compared the results to CP-$\mathcal{MM}$AS. As discussed earlier, this was to determine which implementation is more effective on the hard constrained version of the problem.

The parameter settings for q_0 and ρ were obtained as follows. A problem instance from each machine size was selected and run for 30 minutes with a cross product of values from $q_0 = \{0.3, 0.5, 1.0\}$ and $\rho = \{0.1, 0.01\}$ to get an idea of which parameters were best suited to this problem. Given these tests, $q_0 = 1.0$ and $\rho = 0.1$ were selected for ACS. The was no clear advantage for CP-$\mathcal{MM}$AS and therefore $q_0 = 0.5$ and $\rho = 0.1$ were selected. The CPACO algorithms constructed 10 solutions per iteration and for consistency, θ was set to 10 for CP-Beam-ACS. $\mu = 3.0$ was also found to be the best multiplier after tuning by hand. The number of samples (N^s) was set to 5. This small number was chosen since a call to the solver is relatively small for this problem and a cost effective estimate can be obtained relatively quickly if only 5 samples are computed per variable. 30 runs were conducted for each application of the algorithm for 60 minutes to every instance.

Algorithm 14 Procedure to determine hard due times

1: INPUT: A MMJS instance m
2: initialize solver, $l = 0.0$, status = fail
3: $\bar{d}_i = (|\mathrm{M}| - 1) \times (r_i + p_i), \forall i \in \{1, \ldots, \mathcal{J}\}$
4: $\bar{d}_i \leftarrow \bar{d}_i + (r_j + p_j) \times 0.1 \times (r_j + p_j), \forall i \in \{1, \ldots, \mathcal{J}), \forall j \in \{1, \ldots, PR(\mathcal{J})_i\}$
5: **while** status = fail **do**
6: $\quad \lambda \leftarrow \lambda + 0.1$
7: $\quad \bar{d}_i = d_i, \forall i \in \{1, \ldots, PR(\mathcal{J})\}$
8: $\quad \bar{d}_i \leftarrow \bar{d}_i + \lambda \times (r_j + p_j), \forall i \in \{1, \ldots, \mathcal{J}), \forall j \in \{1, \ldots, PR(\mathcal{J})_i\}$
9: $\quad$ status = Solve(m)
10: **end while**
11: $\lambda \leftarrow \lambda \times k$
12: $\bar{d}_i \leftarrow \bar{d}_i + \lambda \times (r_j + p_j), \forall i \in \{1, \ldots, \mathcal{J}), \forall j \in \{1, \ldots, PR(\mathcal{J})_i\}$
13: OUTPUT: $\hat{d}_i, \forall i \in \{1, \ldots, \mathcal{J}\}$

Hard Due Times

The datasets were obtained from Singh & Ernst (2010). However, these instances did not include hard due times and the strength of the CP model is enhanced when this type of constraint is present in the model. Thus, we incorporate hard due times and determine them as follows.

The problem instance is setup with a constraint solver with all the basic constraints and the following steps are repeated until the solver state returned is feasible (see Algorithm 14). Initially, a job i has its hard due time set to $(|\mathrm{M}| - 1) \times (r_i + p_i)$ (line 3). The idea here is that the multiplier $(|\mathrm{M}| - 1)$ provides a substantial cushion for the job to complete without violating the constraints. This however may not be sufficient for the jobs with precedences and therefore these jobs have their hard due time set to the sum of all preceding jobs release times plus processing times (line 4). At this stage we only modify the hard due times for those jobs with precedences. Specifically, the algorithm iteratively increases the level (λ) until a the solver state returns true. Here, we are sure that the solver starts with a feasible state but this does not imply that a feasible solution exists. Therefore, as a final step we further increase $l \leftarrow l \times k$ at line 11 and reset the hard due times. In this study, we vary k depending on the problem size, i.e., $|\mathcal{M}| \leq 4 \Rightarrow k = 2$, $4 \leq |\mathcal{M}| \leq 9 \Rightarrow k = 3$ or $10 \leq |\mathcal{M}| \leq 12 \Rightarrow k = 4$.

More sophisticated methods may be used to obtain these hard due times, however, the procedure followed here suffices for the purpose of generating hard instances to solve. Moreover, by tweaking the parameter λ we are also able to create problems of varying hardness.

5.2.8 Results

The major results of this chapter considering optimality and feasibility can be summarised as follows.

- CP-ACS and CP-Beam-ACS are the best performing algorithms
  - CP-Beam-ACS is most effective with respect to feasibility, stochastic sampling providing a clear advantage
  - CP-ACS is by a small margin the best performing algorithm where feasibility is found
- ACS is surprisingly effective, but not as competitive as CP-ACS given the same number of labeling steps
- Beam-ACS preforms poorly: stochastic sampling does not provide an advantage without CP

Overall we see that ACS, CP-ACS and CP-Beam-ACS are all effective algorithms when solving the MMJS problem. CP-ACS is effective for the hard constrained version of the problem but is slow due to the solver overhead. This is remedied by CP-Beam-ACS which is as effective as CP-ACS but is able to achieve this in a small amount of time. When considering feasibility, CP-Beam-ACS is the most effective method for solving this problem. In terms of solution quality, ACS is most effective when solutions are easily found, otherwise CP-Beam-ACS is still the most effective algorithm.

The first set of results are presented in Table 5.4. Here we see the comparison between the two variants of ACO and CP-Beam-ACS. The first column specifies the problem instance where the first index is the number of machines and the second index is an identifier. For each algorithm, we report the best (best) solution found across all runs, the average (mean) solution quality across the runs, the associated standard deviations (sd), the number of times the algorithm fails to find feasibility (fail), the number of CP labeling steps (steps) and the number of iterations (iter.) executed by the algorithm in the allowed time. We report CP labeling steps as opposed to total labeling steps since we compare the algorithms in terms of how many calls to the solver are made in the allowed time. The aim will be to determine if a difference in a large number of CP calls is useful in the context of this problem. The best results obtained in a table for an instance (including mean and feasibility) are marked in bold face if they statistically significant at $p = 0.05$.

The results show that CP-ACS is the preferred implementation over CP-$\mathcal{MM}$AS for this problem. This is true when considering feasibility and solution quality.

Table 5.4: Results for CP-ACS CP-$\mathcal{MM}$AS and CP-Beam-ACS on a subset of problems selected from Singh & Ernst (2010). Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

	CP-$\mathcal{MM}$AS						CP-ACS						CP-Beam-ACS					
	best	mean	sd	fail	steps	iter.	best	mean	sd	fail	steps	iter.	best	mean	sd	fail	steps	iter.
3 - 23	158.08	**158.08**	0.00	0.00	9.71E6	2.86E3	158.08	158.15	0.17	0.00	1.08E7	2.47E3	158.08	158.61	0.80	0.00	2.64E6	8.54E2
3 - 53	72.53	72.53	0.00	0.00	1.29E7	1.12E4	72.53	72.53	0.00	0.00	1.76E7	8.84E3	72.53	72.53	0.00	0.00	4.21E6	6.42E1
3 - 5	511.78	511.90	0.39	0.00	9.04E6	8.05E3	511.78	511.96	1.02	0.00	1.11E7	4.45E3	511.78	515.34	2.72	0.00	2.44E6	1.35E3
4 - 28	23.99	25.27	0.92	0.00	1.07E7	9.40E3	23.94	**24.30**	0.95	0.00	1.26E7	7.77E3	23.94	24.45	0.86	0.00	2.55E6	7.20E2
4 - 42	125.37	151.90	10.99	0.00	1.49E7	7.47E3	123.95	144.44	16.23	0.00	1.68E7	9.05E3	123.95	149.78	20.02	0.00	3.22E6	1.02E3
4 - 73	73.30	86.03	10.05	0.00	1.26E7	6.30E3	73.30	79.28	5.88	0.00	1.25E7	7.56E3	73.38	78.11	3.29	0.00	1.82E6	5.52E2
5 - 21	200.61	**201.33**	1.56	0.00	9.07E6	5.66E3	194.31	205.07	4.93	0.00	1.01E7	4.11E3	200.61	209.73	6.03	0.00	1.60E6	5.30E2
5 - 62	311.66	348.50	19.83	0.00	1.01E7	5.59E3	316.73	355.84	29.38	0.00	1.32E7	7.60E3	335.37	357.41	18.37	0.00	1.46E6	3.81E2
5 - 7				1.00						1.00						1.00		
6 - 10	1197.06	1334.96	64.74	0.67	1.16E7	5.28E3	1194.41	1296.24	61.89	0.37	1.03E7	4.75E3	1211.07	1327.93	93.13	**0.00**	7.79E5	2.40E2
6 - 28	258.16	270.49	6.61	0.00	8.56E6	8.18E3	246.61	275.55	13.81	0.00	9.50E6	5.38E3	251.68	274.67	12.93	0.00	1.46E6	5.45E2
6 - 58	333.27	358.54	8.60	0.07	1.29E7	5.82E3	323.83	355.95	16.23	0.00	1.22E7	4.57E3	332.70	364.20	16.37	0.00	1.70E6	4.18E2
7 - 15				1.00						1.00						1.00		
7 - 23	671.39	698.86	14.92	0.00	7.04E6	3.53E3	636.33	688.34	25.96	0.00	9.47E6	4.51E3	626.95	677.38	30.14	0.00	9.21E5	3.29E2
7 - 47				1.00						1.00			1009.81	**1066.33**	62.12	**0.83**	1.40E6	2.45E2
8 - 3				1.00						1.00			1246.56	**1564.81**	213.35	**0.00**	7.39E5	1.66E2
8 - 53	507.26	525.65	11.50	0.00	4.91E6	3.07E3	487.00	530.72	19.47	0.00	5.67E6	3.75E3	495.99	537.55	16.33	0.00	7.58E5	2.83E2
8 - 77	1289.38	1396.30	40.60	0.00	5.40E6	2.41E3	1278.31	1348.06	36.08	0.00	5.75E6	2.50E3	1330.63	1449.28	67.74	0.00	5.95E5	1.97E2
9 - 20	1039.58	1117.15	35.54	0.00	4.50E6	1.67E3	988.60	1057.67	26.83	0.00	4.53E6	2.21E3	1094.99	1213.07	47.43	0.00	5.24E5	1.53E2
9 - 22	1260.51	1337.74	35.52	0.00	4.54E6	2.45E3	1242.08	1282.66	28.56	0.00	4.08E6	2.36E3	1316.80	1417.96	52.13	0.00	5.44E5	1.67E2
9 - 47	1495.98	1579.70	42.32	0.00	3.27E6	1.08E3	1283.00	1354.77	40.21	0.00	3.19E6	2.07E3	1589.74	1673.19	52.68	0.00	3.74E5	7.56E1
10 - 13				1.00						1.00			4608.86	**5036.25**	317.75	**0.13**	7.87E5	8.64E1
10 - 77				1.00						1.00			1835.65	**1941.00**	75.59	**0.13**	1.18E6	1.80E2
10 - 7				1.00						1.00						1.00		
11 - 21	1248.73	1336.53	43.78	0.00	4.74E6	1.20E3	1172.95	**1228.23**	32.31	0.00	4.20E6	2.02E3	1524.44	1609.91	52.58	0.00	4.19E5	5.87E1
11 - 56	2621.59	2927.41	218.78	0.17	3.96E6	9.28E2	2111.05	2440.69	188.67	0.00	5.13E6	2.17E3	2422.26	2663.11	116.35	0.00	3.32E5	4.45E1
11 - 63				1.00			3943.91	3943.91	0.00	0.97	5.68E6	1.52E3	3320.50	3616.50	222.56	**0.43**	8.19E5	9.03E1
12 - 14				1.00			3288.91	3288.91	0.00	0.97	7.47E6	2.07E3	3697.89	4108.67	159.56	**0.03**	5.81E5	6.20E1
12 - 36				1.00						1.00			5324.54	**5647.87**	222.21	**0.00**	3.96E5	2.28E1
12 - 80				1.00			3481.23	3969.04	410.83	0.87	4.77E6	1.63E3	3856.62	4338.52	222.52	**0.03**	4.40E5	3.22E1

Therefore, ACS is used in the implementation of the beam search - CP variant. These results are reported in the last six columns of Table 5.4.

The estimate used for CP-Beam-ACS was stochastic sampling and it is used in a similar manner as it was used in CP-Beam-$\mathcal{MM}$AS for the SMJS problem. The basic idea is start with a partial solution and complete it by sampling the pheromone trails using Equation 5.7 and Equation 5.8. This is done N^s times. The samples provide two estimates to the partial solutions. Firstly, they provide an estimate of the number of violations of the hard due times determined from the complete sequence. These samples are constructed by relaxing the hard due time constraint and the lowest number of violations among the samples is returned as the estimate. The second estimate is the minimum value of the weighted tardiness across the samples. More formally,

$$\phi_i = (min_{j \in N^s}(\nu(\pi_j)), min_{j \in N^s}(f(\pi_j)) \tag{5.11}$$

for the i^{th} variable and $j \in N^s$. We prioritize by minimizing over $\nu(\pi)$ as feasibility is necessary before we can optimize $f(\pi_j)$ as we do with the higher level ACS search. Alternative estimates are analyzed in Chapter 6.

The results here show that CP-Beam-ACS is the best method for feasibility and optimality on several problem instances. Differences between the algorithms are also exaggerated for the larger instances with more machines and jobs. Since both algorithms use CP the improved feasibility can be attributed to the estimate provided by stochastic sampling. It is also worth noting that CP-Beam-ACS compared to CP-ACS completes far fewer iterations. This amounts to fewer pheromone updates and slower algorithm convergence which is traded-off for a more complex single iteration with beam search. Additionally, CP-Beam-ACS completes fewer CP labeling steps. This is consistent with the number of iterations, i.e., for feasible solutions CP-Beam-ACS completes three times the number of CP labeling steps ($\mu = 3.0$) per iteration but CP-ACS on average completes more than three times the number of iterations.

The results are also plotted in Figure 5.6 and Figure 5.7. Figure 5.6 plots the cumulative failures for each instance for ACS, CP-ACS and CP-Beam-ACS. This figure shows that ACS and CP-ACS fail on over 30% of the total number of runs ($> 10/30$) where CP-Beam-ACS only fails on about 15% of the runs. If CP-Beam-ACS fails on every run then so do the other algorithms. Figure 5.7 shows the % difference to the best algorithm in terms of solution quality for all the instances where feasibility is found averaged across machine size. CP-ACS is the best performing algorithm in this regard. However, there are several instances which can not be compared since CP-ACS does not find solutions for these instances.

Table 5.5: Results for SA, ACS and Beam-ACS on the same problem instances. Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

	SA				ACS					Beam-ACS				
	best	mean	sd	fail	best	mean	sd	fail	iter.	best	mean	sd	fail	iter.
3 - 23	158.08	158.08	0.00	0.00	158.08	158.08	0.00	0.00	6.76E3	158.08	158.08	0.00	0.00	2.13E2
3 - 53	72.53	74.31	2.02	0.00	72.53	**72.53**	0.00	0.00	9.81E3	72.53	**72.53**	0.00	0.00	9.37E2
3 - 5	511.78	515.05	4.58	0.00	511.78	**511.78**	0.00	0.00	2.90E3	511.78	**511.78**	0.00	0.00	2.55E2
4 - 28	23.94	**23.94**	0.00	0.00	23.94	24.08	0.48	0.00	1.92E4	23.94	23.98	0.14	0.00	4.65E2
4 - 42				1.00	123.95	146.52	18.48	0.00	2.78E4	123.95	**132.86**	11.92	0.00	1.00E3
4 - 73				1.00	73.38	79.78	7.48	0.00	1.54E4	73.30	**76.83**	2.95	0.00	3.90E2
5 - 21	279.24	305.42	401.42	0.90	194.31	202.40	4.05	0.00	1.83E4	200.41	202.47	2.68	0.00	3.71E2
5 - 62				1.00	307.80	365.55	53.62	0.00	2.95E4	312.47	**340.13**	15.84	0.00	4.05E2
5 - 7				1.00				1.00					1.00	
6 - 10				1.00	1229.13	1264.23	244.31	**0.87**	1.62E4	1243.48	1266.64	635.18	0.93	2.18E2
6 - 28	318.94	394.56	366.94	0.83	249.04	271.85	13.92	0.00	1.72E4	251.02	**269.25**	9.79	0.00	3.36E2
6 - 58				1.00	325.49	359.39	16.63	0.07	2.22E4	321.66	**349.56**	18.50	**0.00**	3.85E2
7 - 15				1.00				1.00					1.00	
7 - 23				1.00	615.26	672.12	22.23	0.00	1.64E4	628.06	**659.15**	23.45	0.00	2.95E2
7 - 47				1.00				1.00					1.00	
8 - 3				1.00	1310.75	1310.75	705.07	0.97	1.14E4	1383.05	1514.91	836.09	**0.87**	1.31E2
8 - 53	592.60	674.02	81.04	0.07	496.66	522.76	15.90	0.00	1.59E4	488.93	**504.04**	8.59	0.00	2.72E2
8 - 77	1274.00	1461.05	173.95	0.33	1259.33	1337.33	36.38	0.00	1.07E4	1283.86	**1334.72**	26.37	0.00	1.77E2
9 - 20	1410.51	1443.04	843.39	0.90	998.94	1045.97	31.34	0.00	7.95E3	1004.52	1071.36	31.86	0.00	1.57E2
9 - 22	1220.75	1262.48	21.35	0.00	1215.56	1270.19	28.47	0.00	8.45E3	1211.87	1244.03	20.29	0.00	1.89E2
9 - 47	1242.53	1310.90	32.72	0.00	1231.44	1309.94	45.20	0.00	7.74E3	1307.13	1373.18	28.72	0.00	8.20E1
10 - 13				1.00				1.00					1.00	
10 - 77				1.00				1.00					1.00	
10 - 7				1.00				1.00					1.00	
11 - 21				1.00	1106.13	**1201.74**	44.63	0.00	1.08E4	1345.47	1420.53	35.22	0.00	7.08E1
11 - 56				1.00	2075.90	**2255.02**	88.46	0.00	7.06E3	2422.06	2533.90	79.88	0.00	4.88E1
11 - 63				1.00				1.00					1.00	
12 - 14				1.00	3039.25	**3113.78**	1029.90	**0.93**	5.39E3				1.00	
12 - 36				1.00				1.00					1.00	
12 - 80				1.00				1.00					1.00	

Experiments were also conducted to determine how simulated annealing, ACS and Beam-ACS without any CP component perform on these problem instances. The result are reported in Table 5.5. Firstly, considering SA, the result show that a large number of problem instances (19/30) are too hard for SA to find feasibility. Even for those problem where feasibility is found SA fails several times (e.g. 6-28 and 9-20). Surprisingly, SA even fails on some of the small instances such as 4-42 where every other algorithm finds feasibility.

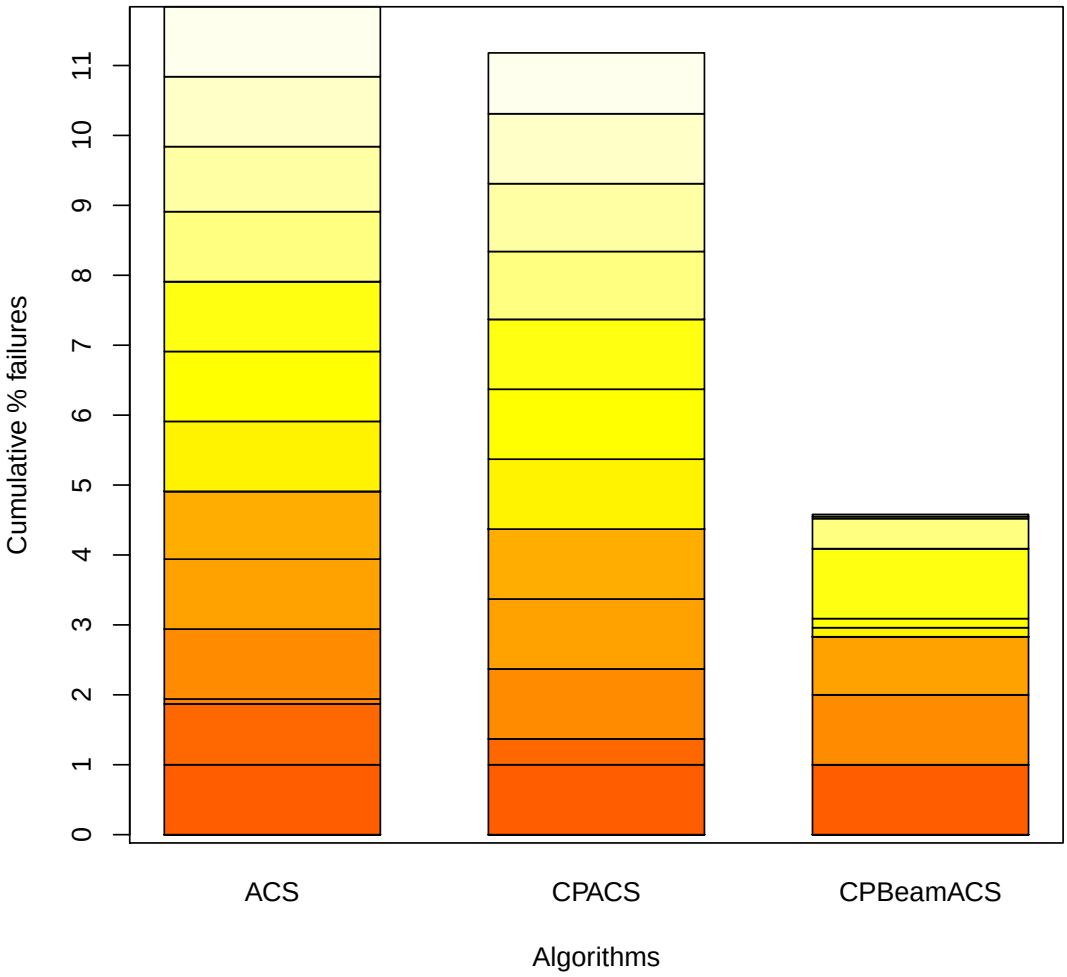


Figure 5.6: Cumulative failure results for ACS, CP-ACS and CP-Beam-ACS. ACS and CP-ACS fail on several instances where their performance is similar with CP-ACS performing slightly better. For each algorithm, each block corresponds to one problem instance where the shading/colour indicates the problem instance and the height of each block indicates the number of failures.

ACS however, performs competitively with CP-ACS . CP-ACS is superior in terms of finding feasibility on some problems (e.g., 6-10 and 7-47) but is surprisingly

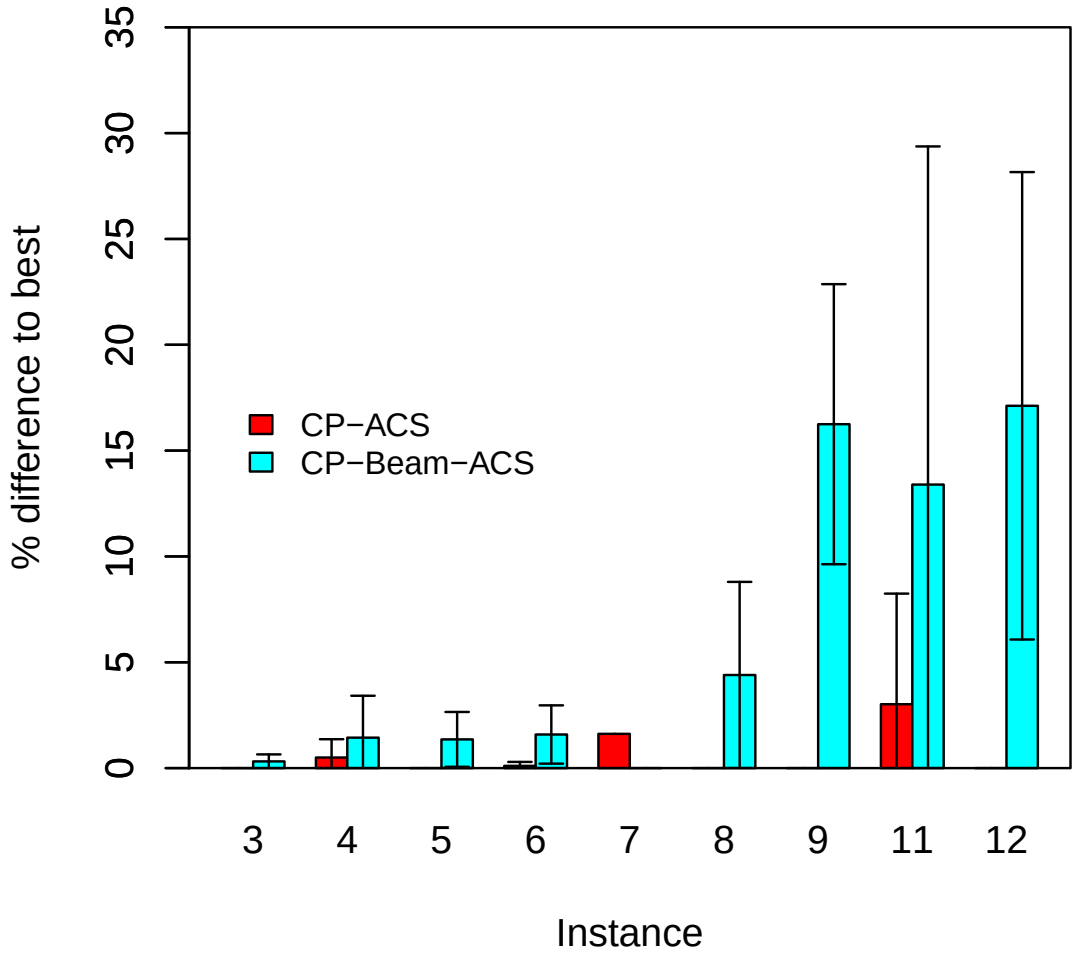


Figure 5.7: Results for CP-ACS and CP-Beam-ACS for those instances where at least one feasible solution was found. For each algorithm, the % difference to the best performing algorithm is shown averaged across each problem size.

worse on one instance - 8-3. For this instance, CP-ACS finds no feasible solution whereas ACS almost always fails but does find feasibility. On close examination it can be seen that ACS performs a much larger number of iterations and the larger number pheromone updates are able to lead the algorithm to feasibility like the CP component of CP-ACS similarly does. Thus, we conducted experiments with ACS and CP-ACS given both algorithms the same number of iterations (3000 iterations). These results are presented in Table 5.6. Here it can be observed that CP-ACS is in fact superior on most problems where feasibility is an issue. Again, ACS find a solution for instance 8-3 on one run while CP-ACS does not. Generally, though, both algorithms mostly fail on this instance. In comparison with the previous study

Table 5.6: ACS and CP-ACS run for 3000 iterations. Statistically significant results ($p = 0.05$) are marked in boldface.

	ACS				CP-ACS			
	best	mean	sd	fail	best	mean	sd	fail
3 - 23	158.08	158.14	0.15	0.00	158.08	**158.08**	0.00	0.00
3 - 53	72.53	72.53	0.00	0.00	72.53	72.53	0.00	0.00
3 - 5	511.78	513.54	2.61	0.00	511.78	512.95	2.25	0.00
4 - 28	23.94	25.79	1.83	0.00	23.94	**24.82**	1.25	0.00
4 - 42	128.65	154.25	40.51	0.37	125.29	157.69	17.02	**0.00**
4 - 73	76.10	85.47	12.51	0.23	73.30	84.16	10.39	**0.00**
5 - 21	200.61	209.16	6.20	0.00	201.27	**206.13**	4.31	0.00
5 - 62	324.11	433.97	99.49	0.40	315.20	**374.85**	52.56	**0.00**
5 - 7				1.00				1.00
6 - 10	1198.59	1198.59	885.03	0.97	1149.24	1280.29	73.56	**0.73**
6 - 28	255.43	286.44	16.60	0.00	252.45	280.70	15.08	0.00
6 - 58	368.30	385.27	78.98	0.57	328.31	352.92	14.36	**0.00**
7 - 15				1.00				1.00
7 - 23	632.56	696.77	31.20	0.03	621.40	**679.75**	26.88	**0.00**
7 - 47				1.00				1.00
8 - 3	1456.15	**1552.45**	626.05	**0.90**				1.00
8 - 53	494.86	531.40	18.40	0.00	498.74	524.02	16.47	0.00
8 - 77	1283.76	1352.90	37.35	0.00	1293.70	1349.90	32.07	0.00
9 - 20	1020.93	1053.39	26.72	0.00	992.55	1047.93	34.52	0.00
9 - 22	1230.37	1280.16	31.29	0.00	1223.70	1276.36	30.53	0.00
9 - 47	1254.09	1323.17	40.33	0.00	1237.84	1309.68	39.15	0.00
10 - 13				1.00				1.00
10 - 77				1.00				1.00
10 - 7				1.00				1.00
11 - 21	1146.78	1229.43	51.16	0.00	1123.84	**1191.76**	46.56	0.00
11 - 56	2129.98	2285.47	129.19	0.10	2155.89	2286.36	110.84	**0.00**
11 - 63				1.00	3121.37	**3121.37**	0.00	**0.97**
12 - 14				1.00	3621.91	**3644.99**	23.08	**0.93**
12 - 36	4944.91	**4944.91**	2002.02	**0.97**				1.00
12 - 80				1.00	3225.40	**3459.17**	151.56	**0.77**

for instances with eight or more machines where CP-ACS complete fewer than 3000 iteration always, CP-Beam-ACS is still more effective than CP-ACS even though the algorithm requires many fewer iterations.

Beam-ACS however, does not provide any feasibility advantage over ACS. The estimate used here again is based on stochastic sampling and here the repeated pheromone updates (measured by number of iterations) are as effective in terms of feasibility. In fact, ACS finds feasible solutions on occasions for instance 12-14 whereas Beam-ACS does not find any feasible solutions for this instance. In terms of solution quality, Beam-ACS is more effective on small-medium size instances while ACS is more effective on the larger instances. However, the differences between the solution qualities are relatively small.

No Learning and no Local Search

Additional experiments were conducted to demonstrate that pheromone learning contributes to the search. For this implementation, the pheromone updates are turned off. This amounts to repeated solution construction from uniform distributions across all jobs at each variable. Following this, local search is applied as described above to the best solution found at each iteration. The results show that ACS without learning is not effective on this problem in terms of feasibility and solution quality.

A final experiment was to determine if the local search does in fact prove useful or not. Here, we turn off the local search altogether when applying the algorithm to all the instances. As the results show, the local search component does indeed contribute to the solution quality. Feasibility is also more easily found for the small instances, however, this effect is not seen with the larger instances.

Table 5.7 presents the results for ACS without learning (ACS-NL) and without local search (ACS-NLS). These results show that learning plays a major part in identifying feasibility and in solution quality. While (ACS-NL) finds feasibility almost always for the smaller instances, the solution quality found is poor compared to ACS with learning.

The results without local search show that feasibility is not significantly affected here. Local search is able to improve feasibility for the smaller instances, e.g., instances 4-42 and 4-73. However, this does not seem to be the case for larger instances (≥ 8 machines). In fact feasibility is found on some instances where feasibility was not found earlier with local search (e.g., 12-36 and 12-80). This can be attributed to the number of pheromone updates due to the number of iterations which increase by a one or two orders of magnitude across all problems. Hence, for larger problem instances, more frequent pheromone updates lead to greater feasibility. In terms of solution quality, the local search component almost always produces improved results (≥ 4 machines).

CP Propagation Costs

In order to further assess the cost of stochastic sampling versus CP solver costs, we examine a single problem instances from each machine size. For each instance, 10 iterations are allowed where the time required for a CP call (i.e., calling the solver and obtaining its state) and the time required for stochastic sampling are measured independently. The average results across the ten runs are reported in Figure 5.8. This figure shows that in general CP calls for this problem are much quicker than stochastic sampling. The cost for CP calls increase by a constant factor with the

Table 5.7: ACS without learning and with local search. Statistically significant results ($p = 0.05$) are marked in boldface.

	ACS-NL					ACS-NLS				
	best	mean	sd	fail	iter.	best	mean	sd	fail	iter.
3 - 23	158.59	159.73	1.03	0.00	4.27E4	158.08	159.97	2.89	0.00	1.63E5
3 - 53	72.53	72.53	0.00	0.00	4.99E3	72.53	72.53	0.00	0.00	3.84E4
3 - 5	533.53	549.79	9.29	0.00	3.19E4	511.78	**527.37**	23.55	0.00	1.11E5
4 - 28	29.35	30.09	0.59	0.00	3.79E4	23.94	**27.07**	2.31	0.00	2.12E5
4 - 42	204.60	268.35	104.92	0.27	2.97E4	125.29	**163.36**	27.65	0.13	3.05E5
4 - 73				1.00	0.00E0	76.10	**88.31**	19.97	0.23	2.66E5
5 - 21	244.86	262.08	8.07	0.00	2.78E4	200.61	**215.67**	8.19	0.00	1.57E5
5 - 62				1.00	0.00E0	325.30	**411.61**	71.18	0.50	1.56E5
5 - 7				1.00	0.00E0				1.00	0.00E0
6 - 10				1.00	0.00E0				1.00	0.00E0
6 - 28	321.60	357.51	15.64	0.00	2.65E4	261.26	**298.42**	28.77	0.00	1.69E5
6 - 58				1.00	0.00E0	355.68	**407.59**	75.88	0.50	1.80E5
7 - 15				1.00	0.00E0				1.00	0.00E0
7 - 23	918.66	1028.90	165.56	0.37	1.45E4	662.58	**728.79**	33.38	0.03	1.47E5
7 - 47				1.00	0.00E0				1.00	0.00E0
8 - 3				1.00	0.00E0	1496.84	**1496.84**	1574.60	0.97	5.64E4
8 - 53	572.17	603.34	12.93	0.00	1.33E4	510.87	**549.34**	23.78	0.00	1.25E5
8 - 77	1610.71	1694.76	53.87	0.00	1.01E4	1324.74	**1410.66**	55.73	0.00	1.23E5
9 - 20	1393.34	1464.79	48.04	0.00	9.79E3	1023.57	**1145.55**	67.44	0.00	1.09E5
9 - 22	1473.97	1501.93	13.84	0.00	6.63E3	1270.71	**1328.89**	30.11	0.00	1.10E5
9 - 47	1660.98	1697.08	19.43	0.00	5.63E3	1313.30	**1446.77**	71.65	0.00	9.32E4
10 - 13				1.00	0.00E0				1.00	0.00E0
10 - 77				1.00	0.00E0				1.00	0.00E0
10 - 7				1.00	0.00E0				1.00	0.00E0
11 - 21	1679.39	1915.76	171.40	0.03	8.49E3	1225.69	**1326.67**	68.97	0.00	1.05E5
11 - 56				1.00	0.00E0	2105.31	**2376.21**	165.86	0.33	7.52E4
11 - 63				1.00	0.00E0				1.00	0.00E0
12 - 14				1.00	0.00E0				1.00	0.00E0
12 - 36				1.00	0.00E0	4421.56	**4670.30**	1092.16	0.90	7.53E4
12 - 80				1.00	0.00E0	3246.14	**3297.06**	649.66	0.93	8.44E4

problem size whereas the stochastic sampling costs increase approximately quadratically as expected. Note that the variation in the costs for stochastic sampling is attributable to the failure to find feasibility for some instances, e.g. 10 machines.

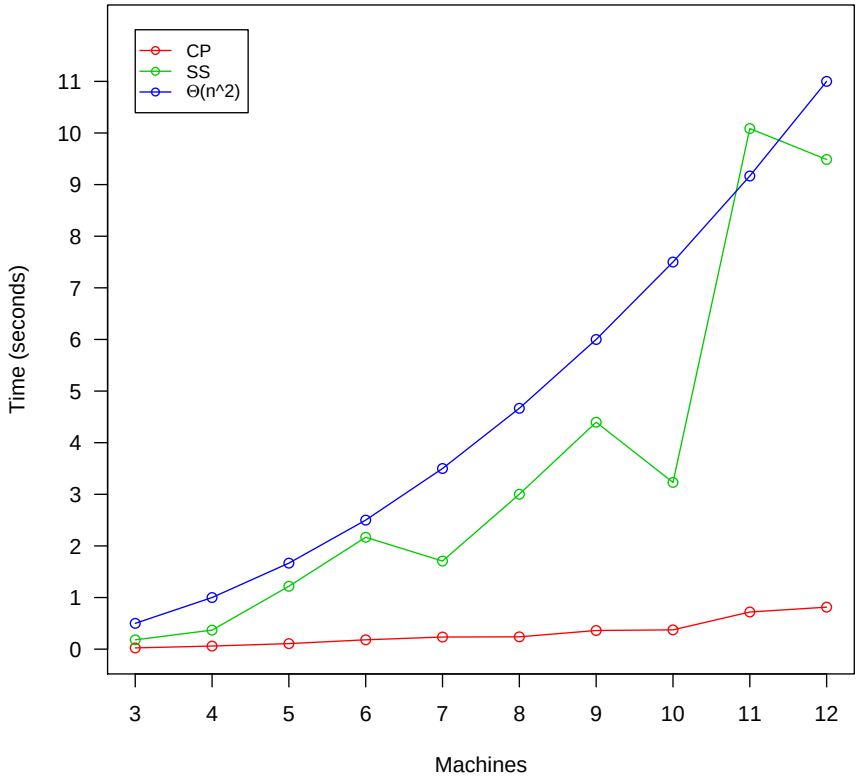


Figure 5.8: This figure shows the time needed (seconds) for CP (CP solver calls) versus SS (stochastic sampling). An $\Theta(n^2)$ trend-line is also shown. A randomly selected instance from each machine size was run for 10 iterations and the average results across the 10 runs are reported here. Stochastic sampling clearly requires a greater amount of time compared to the CP calls and this difference generally increases with machine size. Note that the results here do not require that feasible solutions are found.

5.2.9 Discussion

The results show that CP-Beam-ACS with stochastic sampling is the most effective algorithm finding feasibility on more instances than the other algorithms. If feasibility is found CP-ACS is the preferred algorithm as it is able to find higher quality solutions.

Also seen in the results (especially for large problems) are that the CP labeling steps and iterations completed by CP-Beam-ACS are a lot fewer than CP-ACS. This can be explained by the cost of stochastic sampling $O(n^2 \times m)$ where we have

n variables and m samples. If a call to the CP solver is inexpensive, then as problem instances get larger, the cost of stochastic sampling is going to outweigh the cost of the solver. As we have seen with the SMJS problem, we are effectively trading off CP solving time with stochastic sampling.

The question as to whether this is a useful trade-off arises. Stochastic sampling proves to be effective for MMJS particularly with respect to feasibility. Given that the CP model is relatively simple, we see that the feasibility estimate provide a distinct advantage leading CP-Beam-ACS into feasible regions. This leads to the question of the usefulness of the CP model. CP-ACS outperforms ACS overall in terms of feasibility demonstrating the need for CP. However, stochastic sampling on its own does not provide a feasibility advantage. Thus the best combination in the context of feasibility is CP combined with stochastic sampling.

We considered a relatively simple version of local search here. More complex methods such as variable neighbourhood descent may be more effective than the constantly applying the two neighbourhood moves suggested above. Furthermore, the simulated annealing heuristic may also be useful here. However, the aim here is to examine the effectiveness of parallel solution construction and while the local search may be improved it is independent of the solution construction and is not expected to have any effect in this component of the algorithm (e.g. biasing the solution construction in the beam).

5.2.10 Conclusion

In this study we apply ACS, CP-ACS and CP-Beam-ACS to a resource constrained multiple machine job scheduling problem. We see that CP-Beam-ACS using stochastic sampling is the most effective algorithm here finding feasible solutions where the other algorithms struggle. ACS and CP-ACS are both effective when feasible solutions are found often outperforming CP-Beam-ACS.

As opposed to (Thiruvady et al., 2009) for SMJS, we see here that CP-Beam-ACS provides a feasibility advantage compared to an optimality advantage. This is accounted for by considering that stochastic sampling provides a feasibility estimate and in the absence of a strong CP model (as we see for the current problem) stochastic sampling provides guidance into feasible regions. Also seen here is that there are many fewer iterations conducted by CP-Beam-ACS compared to the other algorithms due to the cost of stochastic sampling. This is reflected in the results of solution quality where if feasible solutions are found CP-ACS is the best algorithm.

This study further validates pheromone model for sequences for such scheduling problems in addition to CP models for the same problems. It is already known that such models have been effective on similar problems independently (den Besten

et al., 2000; Bauer, Bullnheimer, Hartl and Strauss, 2000). Hybridizing these models proves to be useful and furthermore incorporating the beam component further enhances the performance of the algorithm.

The form of local search used here is relatively naïve, i.e., repeated application of random swapping and $\beta-$sampling. We have also shown that the local search is useful. Therefore, a more sophisticated local search, such as variable neighbourhood descent, might provide improved results and this component of the algorithm warrants testing in the future.

5.3 Car Sequencing

The car sequencing problem has been actively researched since the mid 1980s with one the first studies being (Parrello, Kebat and Wos, 1986). The problem requires a number of cars to be scheduled on an assembly line (Dincbus, Simonis and Hentenryck, 1988; Gent, 1998; Hentenryck, Simonis and Dincbus, 1992; Parrello et al., 1986) and this problem has become a benchmark CSP problem (Gent and Walsh, 1999).

Each car requires a number of options (e.g., air conditioning, sunroof, radio, etc.) where each option is installed at a particular station. The stations can only handle a limited number of cars at a time. Hence, cars requiring the same options must be sequenced such that no station's capacity is exceeded. The satisfiability problem of identifying such a sequence is known to be NP-hard (Kis, 2004).

As a pure satisfiability problem, car sequencing has been effectively tackled with constraint and integer programming approaches (Dincbus et al., 1988; Gravel, Gagné and Price, 2004; Hentenryck et al., 1992). In (Dincbus et al., 1988), it was shown that a pure CP approach is very effective on this problem by sequencing up to 200 cars within five minutes, even for those instances with high utilization of options (e.g. 90 %). In (Gravel et al., 2004), an integer programming approach was proposed where the number of violated constraints is minimized subject to a number of linear constraints.

ACO on its own has also been successful with satisfaction problems in the past (Solnon, 2002; Solnon, 2008; Roli, Blum and Dorigo, 2001). These approaches are based on relaxing several constraints during solution construction and minimizing violations as the objective. In (Solnon, 2008), it was shown that ACO on its own can be effective for small instances of car sequencing whereas approaches based on local search are more effective on larger instances. Transferring Meyer & Ernst's (Meyer and Ernst, 2004) approach to satisfaction problems, Kichane *et al.* (Khichane et al., 2008) use a CP approach which is aided by ACO when attempting to find feasibility. They show excellent results on large and hard instances of the problem (Perron and Shaw, 2004).

In this study, we explore an optimization version of the car sequencing problem (Bautista, Pereira and Adenso-Díaz, 2008): feasible solutions to the car sequencing problem must be found and additionally the utilization of options must be modulated within sub-sequences. Note that a pure beam search approach is proposed in (Bautista et al., 2008), but it focuses on feasibility as opposed to the additional objective. We examine CP-Beam-ACO and CP-Beam-$\mathcal{MM}$AS on this optimization problem by also focusing on the second objective. We also consider non-CP approaches (ACO and Beam-ACO (Blum, 2005)) but, as expected, both approaches struggle with obtaining feasibility.

5.3.1 Problem Definition

The formal definition of the car sequencing problem is as follows (Solnon, Cung, Nguyen and Artigues, 2008). A number of cars $\mathcal{V} = \{v_1, \ldots, v_n\}$ and options $\mathcal{O} = \{o_1, \ldots, o_m\}$ are given. Each car requires a number of options specified by $r_{ij} \in \{0, 1\}, \forall i \in \mathcal{V}, j \in \mathcal{O}$ where $r_{ij} = 1$ states that car i requires option j. We are also given $(p_i, q_i), i \in \mathcal{O}$ which impose constraints that require that at most p_i cars may use option o_i in a subsequence of length q_i satisfying the capacity of a station. The cars are grouped into classes $\mathcal{C}$ requiring the same options such that for all $l \in \mathcal{C}$, $i,k \in l$, $j \in \mathcal{O} : r_{ij} = r_{kj}$. We represent a solution to this problem by a sequence π which consists of n variables each of which have domains $\{c_1, \ldots, c_l\}$ such that the option constraints are satisfied.

Like any satisfiability problem, the car sequencing problem can be framed as an optimization problem. Essentially, for constructive methods, the objective is to maximize the number of decision variables assigned such that constraints are satisfied. The search terminates when all decision variables are assigned, i.e. all cars are sequenced. Additionally, departing from satisfiability only, (Bautista et al., 2008) considers further aspects of the problem which can be optimized. They define two different but related measures based on feasibility and how well the options are modulated across sub-sequences.

The first measure is the *upper over-assignment* of a sequence which is the number of times an option appears over the allowed number when the sub-sequence constraint is violated:

$$uoa(\pi) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \times y_{ij}(\pi) \tag{5.12}$$

where

$$y_{ij}(\pi) = max\left[0, -min(j, p_i) + \sum_{k=u_i(j)}^{j} r_{\pi_k i}\right] \tag{5.13}$$

where $u_i(j) = max(1, j+1-q_i)$. The a_{ij} terms can be seen as penalties for violating the constraints. Here, we do not specify a_{ij}, because we treat this component as a hard constraint. Essentially, the sum in the equation above adds up the number of options that are used in a subsequence from $j + 1 - q_i$ to j. Of course, if we have not sequenced q_i cars yet we only consider the smaller size subsequence. $uoa(\pi) = 0$ implies that a feasible solution to the problem has been found since we are only summing over the quantity of the violations. Similarly, *upper under-assignment* of a sequence can be defined as

$$uua(\pi) = \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij} \times z_{ij}(\pi) \tag{5.14}$$

where

$$z_{ij}(\pi) = max\left[0, min(j, p_i) - \sum_{k=u_i(j)}^{j} r_{\pi_k i}\right] \tag{5.15}$$

where u_{ij} is defined earlier. Via b_{ij}, we are able to modulate the usage of options in the sub-sequences. For our experiments, we randomly select $b_{ij} \in (0, 1]$. This in effect imposes preferences on those sub-sequences where all the allowed options should be sequenced or as many of them as possible. Given these two measures, we can define the optimization version of the problem to minimize

$$uoa(\pi) + uua(\pi) \tag{5.16}$$

5.3.2 Methods

We first discuss the ACO variant used in this study and the pheromone models that have been attempted for this problem. From these, we select the most plausible one. A CP model is then suggested based on previous studies and the integration of this model into $\mathcal{MM}$AS is briefly discussed. To determine which $\mathcal{MM}$AS variants are effective and which parameter settings are optimal we make use of the experience of previous studies.

ACO and CP

In their study on CP and $\mathcal{MM}$AS (Khichane et al., 2008) used an implementation of $\mathcal{MM}$AS for the car sequencing problem and examined various pheromone models. Two straightforward models can be defined as follows. The first is a *default* model which constructs a sequence based on the class of cars. That is, a sequence contains as many variables as there are cars and each variable has domain values that belong to the car classes. This model consists of $O(|V| \times |C|)$ pheromone values. The second obvious model, *classes*, is to select a car class for a variable given the car class in the previously assigned variable ($O(|C|^2)$ pheromone values). Finally, (Khichane et al., 2008) explores a third pheromone model, *cars*, which associates trails with every pair of car classes and the number of remaining unassigned cars in each of these classes ($O(|C|^4)$ pheromone values).

In their study, (Khichane et al., 2008) found that their *cars* model was the best performing model by a small margin, despite the large number of pheromones that need to be learned for this model. Surprisingly, the *classes* model was worse than the *default* model. Given these results, we chose to use the *default* model. It is proven to be more effective than the *classes* model and the cars model is considered too large given the large size of the pheromone trails for larger instances with few iterations. This is particularly true for experiments with large instances where a

Algorithm 15 CP-$\mathcal{MM}$AS for the car sequencing problem

```
1: INPUT: A car sequencing instance
2: π^bs := NULL
3: initialize T
4: while termination conditions not satisfied do
5:    S_iter := ∅, S := ∅
6:    for j = 1 to n_ants do
7:       π_j := ConstructSequence()
8:       S_iter := S_iter ∪ {π_j}
9:    end for
10:   S := S ∪ argmax{f(π)|π ∈ S_iter}
11:   π^bs = Update(S)
12:   T = PheromoneUpdate(S)
13: end while
14: OUTPUT: π^bs
```

time limit is given as the terminating criterion and the CP-based algorithms only complete a few iterations or few pheromone updates. All algorithms compared in this study make use of the same pheromone model and potential differences caused by the pheromone models can be expected to have similar effects across all the algorithms.

The CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS algorithms from Chapter 4 are restated for this problem here partly for readability and mainly for subtle differences between the implementations. The high-level $\mathcal{MM}$AS algorithm is presented in Algorithm 15. The pheromone trails for the *default* model consist of pheromones τ_{ij} where i is the position or variable in the sequence (i.e., π_i) and j is a class.[8] The algorithm is a straightforward version of the $\mathcal{MM}$AS algorithm and the details of the specific procedures are discussed below. All parameter settings including initial values are adopted from (Khichane et al., 2008).

ConstructSequence(): The permutation π of the jobs is constructed by selecting a car class for each variable beginning with the first variable π_1. There are two possible solution constructions allowed here based on whether CP is used or not. For the non-CP algorithms, a solution is considered complete when all the variables have car classes assigned to them. For the CP algorithms, variables are bound starting with π_1 and the construction terminates when either all the variables have been assigned or when the partial solution is recognised as infeasible.

Algorithm 16 shows the procedure for building a single solution with CP which may substitute line 7 in Algorithm 15. This procedure incrementally adds components to the sequence π by making sure that the choices are feasible. Between lines 4 and 9 single level back-tracking is used and all possible domain values ($D =$

[8]Note there is no distinction made between cars of the same class in the default model.

Algorithm 16 Construct solutions using CP

```
1: i ← 0, feasible ← true
2: while i ≤ n & feasible do
3:    i ← i + 1
4:    D = domain(π_i)
5:    repeat
6:       j = select_class(D, T)
7:       feasible = post(π_i ← j)
8:       if not(feasible) then post(π_i ≠ j)
9:       D = D \ j
10:   until D ≠ ∅ ∨ feasible
11: end while
12: Return π
```

domain(π_i)) are tested for a variable until a feasible choice is found. If no feasible solution is found a partial solution with not all variables assigned may be returned.

The selection of a car class proceeds in standard ant colony system fashion (procedure select_class(D,τ)). We generate a random number $q \in (0, 1]$ and compare this with a pre-defined parameter q_0. For the i^{th} variable, π_i, if $q < q_0$, a job k for variable i in π is greedily selected according to

$$k = \max_{j \in \mathcal{C}} \tau_{ij} \times \eta_j^{\beta} \tag{5.17}$$

otherwise, $\pi_i = k$ is selected with probability

$$\mathbf{p}(\pi_i = k) = \frac{\tau_{ik} \times \eta_k^{\beta}}{\sum_{j \in \mathcal{C}} \left(\tau_{ij} \times \eta_j^{\beta}\right)} \tag{5.18}$$

where the heuristic factor η_k^{β} for selecting car class k for variable i of permutation π. β is a factor that specifies the relative contribution of the heuristic weight and will be defined in the results section. For η_k, We use the dynamic sum of utilisation (DSU) rates first suggested by Smith (Smith, 1997) and shown to be the best performing heuristic by Gottlieb *et al.* (Gottlieb, Puchta and Solnon, 2003). The idea is to make use of the utilization rate of an option o_i which is defined as the ratio of the number of cars that require o_i to the maximum number of cars that can have o_i while satisfying its capacity constraint:

$$\eta_k = \sum_{o_j \in ro(c)} \frac{rv(o_j, n_j)}{n - i} \tag{5.19}$$

where n_j are the number of cars that require option o_j but have not been sequenced in π yet. The set of options needed by car class c is specified by $ro(c)$. $n - i$ is the

number of variables left to assign in the partial solution. $rv(o_j, n_j)$ is the utilization rate of option o_j, which is the ratio of the total number of cars n_j requiring option o_j to the maximum number of variables for installing the option satisfying the capacity of a station (Khichane et al., 2008). It is computed as follows:

$$rv(o_j, n_j) = \begin{cases} \frac{q_j \times n_j}{p_j} - (q_j - p_j) & if\ n_j \% p_j = 0 \\ \frac{q_j \times (n_j - n_j \% p_j)}{p_j} + n_j \% p_j & otherwise \end{cases} \tag{5.20}$$

$\pi^{bs} = \mathsf{Update}(S)$: This procedure sets π^{bs} to the best solution in S. S consists of the best solutions from the current iteration. The cost of a solution is based on two measures $f(\pi_i)$, the number of variables assigned, and $uua(\pi_i)$. We first maximize over $f()$ and the minimize over $uua()$. Therefore, if $\pi_i \in S$ satisfies $f(\pi_i) > f(\pi^{bs})$ or $f(\pi_i) = f(\pi^{bs}) \wedge uua(\pi_i) < uua(\pi^{bs})$ then π^{bs} is set to π_i. Note that the solutions in S have the same number of variables assigned but may have different uua costs.

$\mathcal{T} = \mathsf{PheromoneUpdate}(\pi^{bs})$: This algorithm updates all the solutions that resulted in the best cost for the current iteration. All components (i, j) appearing in the list of best solutions are used to update the corresponding components in the pheromone matrix:

$$\tau_{ij} = \tau_{ij} \times \rho + \delta_S \tag{5.21}$$

where $\delta_S = 1/(1 + f(\pi^{bs}) - f(\pi^{ib}))$. ρ is the learning rate which specifies how quickly the pheromones converge to the best solution and is set to be 0.02 for this study. Here, the pheromones in the CP based algorithms are also rewarded.

The Constraint Programming Model

Since Dincbus *et al.* (1988) applied CP to car sequencing, it has been a benchmark problem for CP solvers. Now various CP solvers include the high level *sequence* constraint with various filtering algorithms for this constraint being proposed (Regin and Puget, 1997; van Hoeve, Pesant, Rousseau and Sabharwal, 2006).

The CP model for this problem make use of two high level constraints. The sequence constraint $sequence(\pi, s, q, l, h)$:
$\bigwedge_{i=0}^{|\pi|-q} among(\langle \pi_i, \ldots, \pi_{i+q-1} \rangle, s, l, h)$ (Gecode, 2010). The $among(\cdot)$ constraint requires that the number of occurrences of $\pi_j = s$ is less than u and greater than l. Using the $among(\cdot)$ constraint, the sequence constraint enforces this constraint for every subsequence of π of size q.

The second high-level constraint used is $count(\pi, |c_l|)$ which specifies that the number of cars of class c_l in π must equal $|c_l|$. By combining these constraints, the CP model is able to achieve strong propagation. Furthermore, these constraints are often solved very quickly. Details will be provided in the results section.

Algorithm 17 Probabilistic Beam Search with CP

1: INPUT: $(\theta, \mu, \mathcal{T})$
2: $B_0 = \{\pi_1 = (), \ldots, \pi_\theta = ()\}$
3: $t = 0$
4: **while** $t < n$ **and** $|B_t| > 0$ **do**
5: **for** $i \in B_t$ **do**
6: $k \leftarrow 0$, $D = \text{domain}(\pi_t^i)$
7: **while** $k < \mu \wedge D \neq \emptyset$ **do**
8: $\hat{\pi} = \pi_i$
9: $j = \text{select\_class}(D,\tau)$
10: $feasible = \text{post}(\pi_t^i \leftarrow j)$
11: **if** ($feasible$) **then** $B_{t+1} = B_{t+1} \cup \pi^k$
12: $k \leftarrow k + 1$, $D = D \setminus j$
13: **end while**
14: **end for**
15: $B_{t+1} = \mathsf{Reduce}(B_{t+1}, \theta)$
16: $t \leftarrow t + 1$
17: **end while**
18: OUTPUT: $\text{argmax}\{f(\pi) \mid \pi \in B_{n-1}\}$

CP-Beam-$\mathcal{MM}$AS

CP-Beam-$\mathcal{MM}$AS is implemented in a similar fashion to the implementation seen in Chapter 4. Algorithm 17 replaces line 7 in Algorithm 15. Here, we start with a beam B_t with θ (initially empty) solutions. For each solution in the beam, μ feasible extensions are constructed and added to B_{t+1}. At this stage, B_{t+1} contains $\theta \times \mu$ partial solutions if these many solutions were found. From among these, we reduce the number of solutions to θ (line 15) using a cost estimate. Here, a cost estimate is computed for each partial solution in B_{t+1}. From these solutions, the best θ solutions are selected in a greedy fashion. The procedure concludes by returning the solution with the best quality.

The estimate used for the beam component of CP-Beam-ACO is analysed later. Here, we briefly describe stochastic sampling adapted for this problem. The basic idea is to consider a partial solution and to complete the solution in plain ACO manner by disregarding the constraints on the options. We obtain a complete solution which has *uoa* and *uua* cost. Given a number of these samples for each solution, the best *uoa* and *uua* costs can be used as an estimate of the true cost for the non-relaxed solution. In (Thiruvady et al., 2009) it was shown that estimates obtained in such a manner can be effective and sometimes even more effective than problem specific lower bounds.

5.3.3 Experimental Setting

Results for experiments with $\mathcal{MM}$AS, CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS were conducted and are reported here. The parameters were determined from (Khichane et al., 2008) for the $\mathcal{MM}$AS and CP-$\mathcal{MM}$AS algorithms and (Bautista et al., 2008) for CP-Beam-$\mathcal{MM}$AS. For all algorithms, $\rho = 0.02$ and $\beta = 6.0$. For beam search, (Bautista et al., 2008) showed that using beam widths of size 25 were effective on the problem and generally improving results were obtained with increasing beam widths as expected. However, their algorithm was run for a single iteration without any learning component. In order to allow for a large number of pheromone updates or effective learning, we chose $n_a = 10$ for CP-$\mathcal{MM}$AS and $\theta = 10, \mu = 3.0$ for CP-Beam-$\mathcal{MM}$AS. In initial experiments we tested values of $\mu = \{2.0, 3.0, 4.0, 5.0\}$ and found that $\mu = 3.0$ was slightly improved compared to $\mu = 2.0$ and $\mu = 4.0$. However, $\mu = 5.0$ was significantly worse. We used 10 samples obtain the estimate for stochastic sampling.

Khichane *et al.* (2008) (Khichane et al., 2008) conducted studies on various datasets from the CSP library (Gent and Walsh, 1999). They found that most instances were easily solved by CP-$\mathcal{MM}$AS and focused their study on the harder datasets used by (Perron and Shaw, 2004). Therefore, in this study we consider instances only from this harder dataset. Furthermore, we consider a subset of instances, six each from instances with 100, 500 and 600 cars and conduct 30 runs per instance. Each run was given 15 hours of execution time and all experiments were conducted on the Monash Sun Grid and the Enterprisegrid using Nimrod (Abramson et al., 2000).

5.3.4 Results

We have tested $\mathcal{MM}$AS, Beam-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS. The major results of this chapter can be summarised as follows.

- CP-Beam-$\mathcal{MM}$AS is the best performing algorithm and is superior to CP-$\mathcal{MM}$AS on 15 out of 18 instances. CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS perform equally with respect to feasibility with CP-$\mathcal{MM}$AS having a slight advantage

- $\mathcal{MM}$AS performs poorly and never finds feasibility

- Two different implementations of Beam-$\mathcal{MM}$AS were attempted. Feasibility was never found with Beam-$\mathcal{MM}$AS and 3 out of 18 instances were solved by Beam-ACO(*uoa*)

Overall, CP-Beam-$\mathcal{MM}$AS is the best performing algorithm for the CS problem inheriting the feasibility characteristics of CP-$\mathcal{MM}$AS and easily outperforming it in terms of optimality.

Table 5.8 shows results for CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS on the problem instances from (Perron and Shaw, 2004). CP-Beam-$\mathcal{MM}$AS is the CP-Beam-ACO version using stochastic sampling to obtain estimates for the beam component. The first column specifies the instance with the first index being the number of cars to sequence and the second index being an identifier. The next six columns show the results for CP-$\mathcal{MM}$AS. The second column shows the best result (number of variables successfully assigned) obtained in terms of feasibility and the third column are the average results across the 30 runs. *uua* is the average *uua* value obtained for the instance and the next column are the associated standard deviations. The next column specifies the CP assignments performed during the execution of the algorithm. *iter* stands for the average of the total number of iterations conducted by the algorithm. The next column specifies the number of runs where the algorithm failed for each instance. The next seven columns are the same results corresponding to CP-Beam-$\mathcal{MM}$AS.

In the time allowed both algorithms find feasibility or come very close (a sequence size that is close to that of the number of cars that need to be sequenced) to finding feasibility on all problem instances. Except for 500-27 and 500-88, CP-$\mathcal{MM}$AS finds feasibility at least on one of the 30 runs. Additionally, CP-Beam-$\mathcal{MM}$AS fails on 100-82. In general CP-$\mathcal{MM}$AS is slightly more effective in terms of feasibility.

The next comparison is of the *uua* values. The values are only considered on the cases where feasibility was found. For example, feasibility for a single run for 100-82 was found with CP-$\mathcal{MM}$AS and only this value is considered for the *uua* average value. The statistically significant results are marked in bold face. Except for one instance (100-82) where CP-Beam-$\mathcal{MM}$AS does not find feasibility, CP-Beam-$\mathcal{MM}$AS outperforms CP-$\mathcal{MM}$AS. Hence, if feasibility is found CP-Beam-$\mathcal{MM}$AS is the preferred algorithm for the optimization version of the problem.

In terms of variable assignments, CP-Beam-$\mathcal{MM}$AS performs significantly more of these compared to CP-$\mathcal{MM}$AS. This is expected since CP-Beam-$\mathcal{MM}$AS performs many more "cheap"[9] assignments when the last few variables need to be assigned. In fact this was an aim of the algorithm design for CP-Beam-$\mathcal{MM}$AS. There is no significant difference in iterations except for CP-$\mathcal{MM}$AS usually performing a few more iterations. In the situation where few feasible solutions are found per iteration, an algorithm will usually complete more iterations, given a time limit, compared to the situation where feasible solutions are always found.

[9]In general, when assigning a variable, CP propagation costs are expensive. However, when relatively few variables are left to label, the propagation costs are relatively small.

Table 5.8: CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS results for selected instances from Perron & Shaw (2004). Statistically significant differences ($p = 0.05$) in *uoa* and *uua* are marked in bold face.

	CP-$\mathcal{MM}$AS							CP-Beam-$\mathcal{MM}$AS						
	best	mean	*uua*	sd	CP steps	iter	fail	best	mean	*uua*	sd	CP steps	iter	fail
100 - 22	100	100.0	274.2	1.44	7.79E+006	1263.6	0	100	100.0	**268.95**	1.27	1.16E+007	1328.7	0
100 - 35	100	100.0	238.2	1.83	7.57E+006	1274.8	0	100	100.0	**231.16**	1.72	1.01E+007	1157.3	0
100 - 64	100	100.0	262.8	1.88	8.22E+006	1458.5	0	100	100.0	**258.35**	1.75	1.09E+007	1335.9	0
100 - 77	100	100.0	188.2	0.93	7.71E+006	1244.4	0	100	100.0	**181.37**	1.53	1.10E+007	1224.7	0
100 - 82	100	98.1	256.3	0.00	8.54E+006	1293.8	29	98	97.3			1.09E+007	1239.8	30
100 - 94	100	100.0	192.7	1.90	7.53E+006	1479.8	0	100	100.0	**183.56**	1.46	1.08E+007	1411.5	0
300 - 8	300	296.9	633.2	3.67	1.44E+006	79.0	27	300	296.9	**614.42**	7.04	1.81E+006	69.7	28
300 - 14	300	300.0	837.9	3.53	1.36E+006	74.4	0	300	300.0	**818.12**	3.66	1.98E+006	75.8	0
300 - 53	300	300.0	697.8	2.21	1.41E+006	74.5	0	300	300.0	**686.07**	1.92	1.63E+006	63.9	0
300 - 56	300	300.0	550.6	2.55	1.52E+006	78.0	0	300	300.0	**534.74**	3.27	2.18E+006	79.7	0
300 - 62	300	300.0	860.0	4.49	1.62E+006	77.6	0	300	299.8	**838.84**	5.24	2.00E+006	71.3	2
300 - 78	300	298.7	626.3	3.91	1.57E+006	75.7	17	300	298.9	**610.70**	6.34	1.82E+006	64.6	16
500 - 14	500	500.0	708.1	2.47	6.40E+005	20.5	0	500	500.0	**687.00**	4.10	9.30E+005	21.2	0
500 - 27	486	482.5			5.89E+005	17.3	30	483	478.8			8.75E+005	20.2	30
500 - 65	500	497.1	804.1	0.00	6.78E+005	20.8	29	500	496.9	**783.42**	0.00	7.77E+005	16.6	29
500 - 74	500	500.0	802.3	6.31	7.18E+005	22.5	0	500	499.9	**776.93**	6.87	9.85E+005	21.4	2
500 - 79	500	500.0	1521.3	4.54	7.11E+005	21.0	0	500	500.0	**1474.83**	5.25	8.93E+005	19.3	0
500 - 88	496	494.4			5.97E+005	19.7	30	495	493.4			8.14E+005	18.8	30

This shows that CP-$\mathcal{MM}$AS may complete fewer close to feasible solutions. With CP-Beam-$\mathcal{MM}$AS other partial solutions may still be in the beam if solutions are rendered infeasible causing an iteration to last longer. This effect may be observed on instance 500-27 where CP-Beam-$\mathcal{MM}$AS on average assigns approximately four fewer variables and hence completes at least three more iterations.

Given the above results, CP-Beam-$\mathcal{MM}$AS is the best choice. However, we used stochastic sampling in this implementation and it is worth examining whether known bounds perform as well or better when implemented with CP-Beam-ACO. In Table 5.9 we show the results for CP-Beam-ACO with bounds based on the *uoa* and *uua* measures. The costs are computed as follows. The *uoa* estimate is computed as the sum of the partial solution's current *uoa* cost and the *uoa* bound. Similarly, the *uua* estimate is the sum of the current *uua* cost and the *uua* bound. The *uoa* & *uua* estimate is the sum of the *uoa* and *uua* estimates. For full details of these bounds see (Bautista et al., 2008).

Table 5.9: Feasibility results for experiments with bounds for the beam search component in CP-Beam-$\mathcal{MM}$AS.

	uoa & *uua*		*uoa*		*uua*	
	best	mean	best	mean	best	mean
100 - 22	100	100.0	100	100.0	100	100.0
100 - 35	100	100.0	100	100.0	100	100.0
100 - 64	100	100.0	100	100.0	100	100.0
100 - 77	100	100.0	100	100.0	100	100.0
100 - 82	98	97.8	100	98.3	100	98.1
100 - 94	100	100.0	100	100.0	100	100.0
300 - 8	298	294.3	300	296.5	300	296.7
300 - 14	300	300.0	300	300.0	300	300.0
300 - 53	300	300.0	300	300.0	300	300.0
300 - 56	300	300.0	300	300.0	300	300.0
300 - 62	300	299.9	300	300.0	300	300.0
300 - 78	300	296.9	300	298.4	300	298.4
500 - 14	500	500.0	500	500.0	500	500.0
500 - 27	483	480.2	487	482.7	484	482.6
500 - 65	498	494.3	500	496.8	500	496.8
500 - 74	500	498.4	500	500.0	500	499.9
500 - 79	500	500.0	500	500.0	500	500.0
500 - 88	495	493.0	495	494.3	496	494.3

Table 5.9 focuses on feasibility. CP-Beam-ACO with *uoa* or *uua* perform well. The *uoa* estimate is expected to be effective since *uoa* is a cumulative measure of how many option we are over the allowed option in the current sequence. Overall, the

uoa estimate is marginally more effective than *uua* and significantly more effective than the combination of the *uoa* and *uua* estimates. We also point out here that there were no significant differences in terms of the *uua* measure.

Given these results, we now compare CP-Beam-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS(*uoa*) in terms of optimality of *uua*(). We only consider those instances where feasibility was found by both algorithms (see Table 5.10). These results show that when feasibility is found, CP-Beam-$\mathcal{MM}$AS is the best performing algorithm across all instances. See also Figure 5.9 which shows the % difference of CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS(*uoa*) to CP-Beam-$\mathcal{MM}$AS. Hence for the optimizaiton version of the problem, CP-Beam-$\mathcal{MM}$AS is the preferred algorithm.

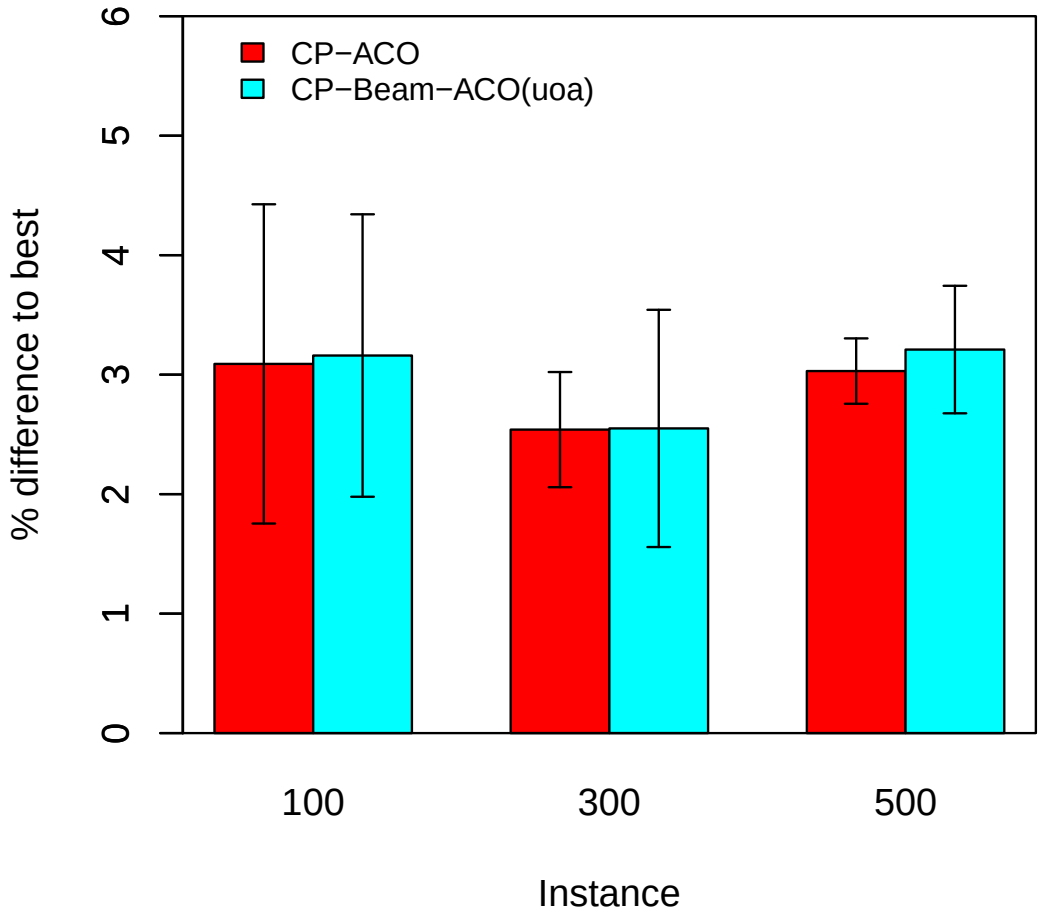


Figure 5.9: The uua results for CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS(*uoa*); For each algorithm, the % difference to CP-Beam-$\mathcal{MM}$AS is shown averaged across each problem size.

Table 5.10: Comparison between CP-Beam-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS(uoa) where feasibility is found.

	CP-Beam-$\mathcal{MM}$AS(uoa)		CP-Beam-$\mathcal{MM}$AS	
	uua	sd	uua	sd
100 - 22	274.23	1.71	**268.95**	1.27
100 - 35	238.19	1.79	**231.16**	1.72
100 - 64	262.83	1.59	**258.35**	1.75
100 - 77	187.67	1.17	**181.37**	1.53
100 - 94	192.63	1.37	**183.56**	1.46
300 - 8	616.82	7.13	614.42	7.04
300 - 14	838.73	2.73	**818.12**	3.66
300 - 53	697.77	1.56	**686.07**	1.92
300 - 56	550.62	2.69	**534.74**	3.27
300 - 62	858.84	4.9	**838.84**	5.24
300 - 78	626.24	8.58	**610.70**	6.34
500 - 14	708.6	2.77	**687.00**	4.10
500 - 65	797.85	0.16	**783.42**	0.00
500 - 74	803.93	5.19	**776.93**	6.87
500 - 79	1522.08	4.18	**1474.83**	5.25

Investigating CP Influence

We now examine the contribution of the CP component. To do so we eliminate any CP from the hybrids, obtaining $\mathcal{MM}$AS and Beam-$\mathcal{MM}$AS. Without the CP support, neither of these algorithms finds feasibility on any instance on any run (see Table 5.11). Note that Beam-$\mathcal{MM}$AS uses stochastic sampling as the estimate and as we had seen in Table 5.8, this estimate provides no feasibility advantage over plain CP-$\mathcal{MM}$AS. Therefore, we cannot expect any advantage over CP-ACO using this estimate with Beam-$\mathcal{MM}$AS.

Beam search using the uoa bound guides the algorithm towards feasibility as seen in (Bautista et al., 2008). Thus Beam-ACO(uoa) may be better placed to find feasibility and we examine this algorithm here. We retain the previous parameter settings of $\theta = 10$ and $\mu = 3$. Table 5.12 shows that CP-Beam-$\mathcal{MM}$AS(uoa) is more effective when considering feasibility.[10] There are three instances where Beam-ACO(uoa) finds feasibility although infrequently for two instances. Comparing these algorithms clearly shows that the CP component provides a big advantage over only using the bound.

[10]Note that Beam-ACO(uoa) does not use the DSU static heuristic.

Table 5.11: $\mathcal{MM}$AS and Beam-$\mathcal{MM}$AS results for selected instances from Perron & Shaw (2004). Statistically significant differences ($p = 0.05$) in *uoa* and *uua* are marked in bold face.

	$\mathcal{MM}$AS				Beam-$\mathcal{MM}$AS			
	uoa	sd	*uua*	sd	*uoa*	sd	*uua*	sd
100 - 22	71.60	0.59	357.45	2.73	**66.3**	0.90	**333.64**	2.59
100 - 35	59.82	0.74	316.56	2.72	**56.39**	1.06	**292.87**	2.69
100 - 64	84.00	0.45	343.78	5.65	**80.02**	1.23	**318.55**	2.40
100 - 77	82.60	0.41	278.66	2.39	**78.36**	0.84	**253.61**	1.75
100 - 82	105.38	0.61	346.17	3.08	**101.60**	1.30	**324.38**	2.70
100 - 94	53.02	0.69	271.72	3.89	**49.51**	1.00	**244.52**	2.69
300 - 8	200.63	1.88	839.32	7.35	**178.08**	2.41	**763.90**	6.74
300 - 14	237.81	1.60	1092.65	9.35	**212.90**	2.59	**1012.11**	5.47
300 - 53	184.42	1.88	913.31	7.86	**165.45**	2.45	**844.57**	6.35
300 - 56	257.62	1.19	826.49	7.14	**232.49**	2.90	**755.00**	5.48
300 - 62	224.71	2.37	1104.79	6.90	**196.79**	2.16	**1015.31**	6.70
300 - 78	244.55	1.98	871.23	7.22	**214.78**	2.44	**790.57**	5.54
500 - 14	411.56	3.36	1164.63	9.69	**353.84**	3.25	**1028.22**	7.25
500 - 27	621.11	2.91	2246.40	13.01	**572.28**	3.95	**2096.81**	7.97
500 - 65	498.01	1.91	1294.92	9.18	**452.52**	3.91	**1181.95**	6.85
500 - 74	428.69	2.38	1264.26	10.28	**386.19**	3.40	**1147.23**	8.97
500 - 79	530.75	3.41	2070.34	11.31	**473.68**	4.11	**1930.65**	9.43
500 - 88	435.94	2.13	1681.00	10.06	**391.54**	3.58	**1554.87**	8.76

While Beam-ACO(*uoa*) is able to identify feasibility on three problems, its performance compared to CP-Beam-$\mathcal{MM}$AS(*uoa*) is poor across all the instances. Therefore, we conclude that the *uoa* bound is only effective if large beam widths may be used and this is infeasible for use with CP given the memory requirements.[11] Comparing with CP-Beam-$\mathcal{MM}$AS in Table 5.8, we see that CP-Beam-$\mathcal{MM}$AS is more effective in terms of feasibility. Considering uua for the three instances where feasibility is found, Beam-ACO(*uoa*) is very close to CP-Beam-$\mathcal{MM}$AS and is slightly more effective on two instances.

Table 5.12: Comparison between Beam-ACO(*uoa*) and CP-Beam-$\mathcal{MM}$AS(*uoa*).

	Beam-ACO(*uoa*)			CP-Beam-$\mathcal{MM}$AS(*uoa*)		
	uua	sd	fail	uua	sd	fail
100 - 22	0		30	274.23	1.71	0
100 - 35	0		30	238.19	1.79	0
100 - 64	0		30	262.83	1.59	0
100 - 77	181.21	1.78	0	187.67	1.17	0
100 - 82	0		30	253.75	1.54	25
100 - 94	200.18	2.32	27	192.63	1.37	0
300 - 8	0		30	616.82	7.13	27
300 - 14	0		30	838.73	2.73	0
300 - 53	0		30	697.77	1.56	0
300 - 56	0		30	550.62	2.69	0
300 - 62	0		30	858.84	4.9	0
300 - 78	0		30	626.24	8.58	22
500 - 14	676.46	4.94	22	708.6	2.77	0
500 - 27	0		30			30
500 - 65	0		30	797.85	0.16	28
500 - 74	0		30	803.93	5.19	0
500 - 79	0		30	1522.08	4.18	0
500 - 88	0		30			30

Comparison with Beam Search

The final experiment conducted here is to compare Beam-ACO(*uoa*) with (Bautista et al., 2008). We ran our algorithm (disregarding the DSU heuristic) on the same instances. Table 5.13 shows these results for runs of duration 10 minutes. The parameters of our algorithm were set as $\theta = 50$ and $\mu = 20$. The $\mathcal{MM}$AS parameters were set as above. It is not clear what value for μ was used by (Bautista et al., 2008) and here we show that with a reasonable value we achieve results close to that of

[11] We had available 4GB memory and attempted $\theta = 50$. However, we ran out of memory on all instances.

this study. Furthermore, our algorithm including an iterative learning component requires a a smaller value of μ in order to allow for a larger number of pheromone updates. These results shows that overall the original beam search algorithm is marginally superior in terms of feasibility. However, it only solves three instances in all, which is a small proportion considering that the p200, p300 and p400 datasets consist of 10 instances each and none of them are solved. The largest difference is seen for p400. Here, the difference is about 13 units but when considering that we are sequencing 400 cars, this is a relatively small. Thus, if we consider the constraints as hard, pure beam search is really not an option.

Table 5.13: Beam-ACO(*uoa*) repeated for instances from Bautista (2008). The DSU heuristic is not used in these implementations.

	Beam (Bautista et al., 2008)		Beam-ACO(*uoa*)			
	uoa	*uua*	*uoa*	sd	*uua*	sd
10 - 93	3		5.3	0.7	34.4	3.0
16 - 81	1		1.9	0.5	55.5	2.8
19 - 71	3		2.8	0.4	44.8	3.7
21 - 90	4		3.0	0.6	71.3	1.5
26 - 82	0		3.2	0.7	67.0	2.1
36 - 92	2		4.0	1.0	55.7	3.2
41 - 66	0		1.9	0.7	111.6	0.8
4 - 72	0		2.1	0.8	70.4	2.9
6 - 76	6		5.6	0.5	115.3	0.5
p200	10.1	138.2	12.4	1.1	134.5	2.8
p300	16.0	176.0	22.1	1.7	169.2	3.7
p400	11.7	325.1	24.6	2.2	283.9	4.1

CP Propagation Costs

We further assess the cost of stochastic sampling versus CP solver costs. 5 instances were chosen from each problem size, 100-77, 200-3, 300-8, 400-8 and 500-14.[12] For each instance, 10 iterations are allowed where the time required to for a CP call (i.e., calling the solver and obtaining its state) and the time required for stochastic sampling are measured independently. The average results across the ten runs are reported in Figure 5.10. This figure shows that in general CP calls are relatively quick and for 200 cars or less the CP calls are more efficient than stochastic sampling. However, for 300 cars and above, the CP solver costs increase dramatically and stochastic sampling becomes increasingly cheaper. The cost for CP calls increase

[12]The instances with 200 and 400 cars were obtained from p200 and p400 instances used by (Bautista et al., 2008)

exponentially with the problem size whereas the stochastic sampling costs increase approximately quadratically as expected.

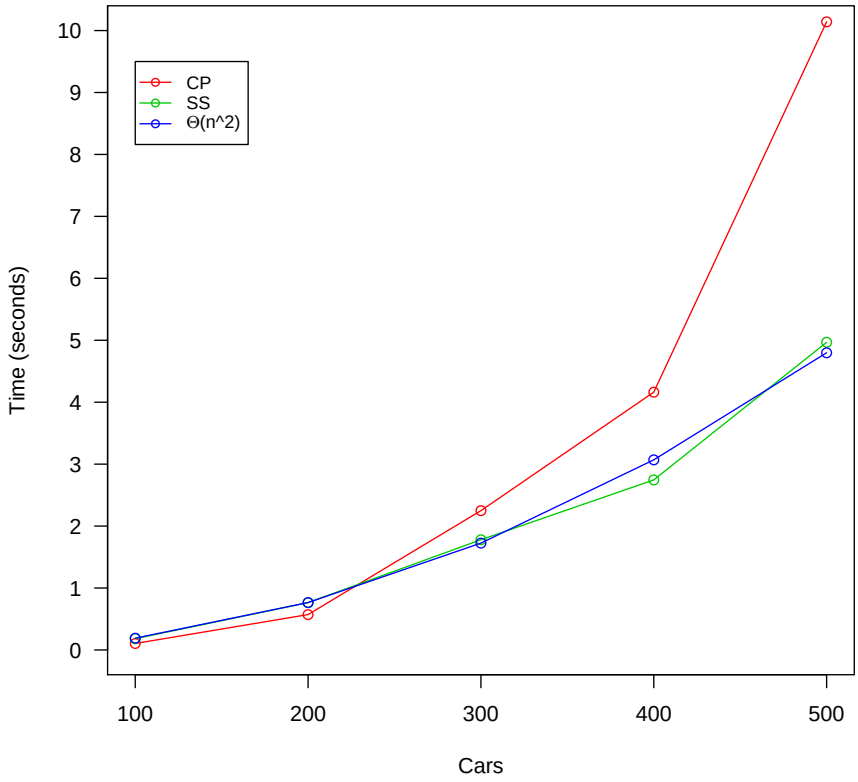


Figure 5.10: This figure shows the time needed (seconds) for CP (CP solver calls) versus SS (stochastic sampling). An $\Theta(n^2)$ trend-line is also shown. A randomly selected instance from each machine size was run for 10 iterations and the average results across the 10 runs are reported here. The CP calls clearly require a greater amount of time compared to stochastic sampling and this difference generally increases with the number of cars. Note that the results here do not require that feasible solutions are found.

5.3.5 Discussion

The results are summarised as follows. $\mathcal{MM}$AS and Beam-$\mathcal{MM}$AS struggle and never find feasibility. Both CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS are able to effectively find feasibility for the optimization version of the car sequencing problem (see Figure 5.11). CP-$\mathcal{MM}$AS is marginally more effective in terms of feasibility. The CP algorithms easily outperform Beam-$\mathcal{MM}$AS with a large beam width despite the fact that the latter uses better bounds. Unfortunately, this better bound (*uoa*) becomes unusable with CP due to the large beam width requirements.

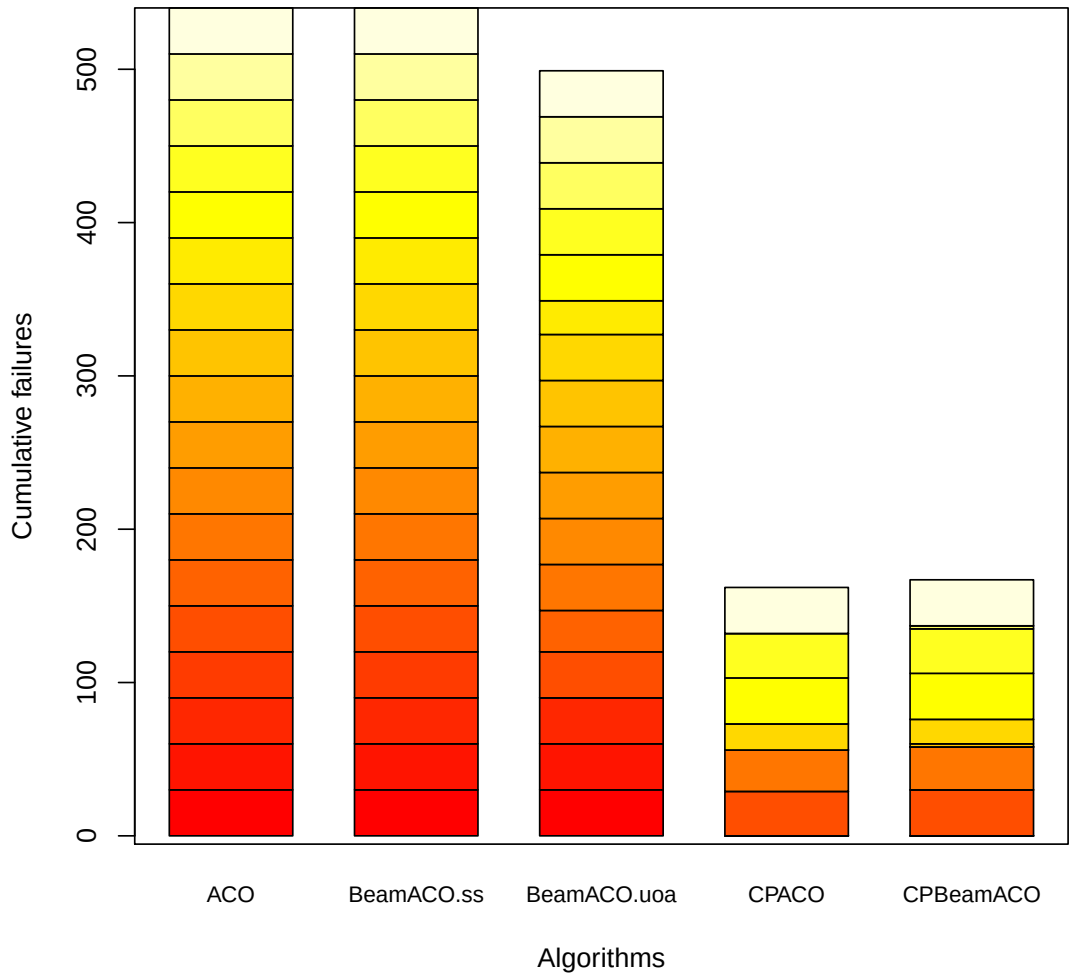


Figure 5.11: Cumulative failure results for $\mathcal{MM}$AS, Beam-$\mathcal{MM}$AS, Beam-ACO(*uoa*), CP-$\mathcal{MM}$AS and CP-Beam-$\mathcal{MM}$AS; For each algorithm, each block corresponds to one problem instance where the shading/colour indicates the problem instance and the height of each block indicates the number of failures.

In terms of optimization performance, CP-Beam-$\mathcal{MM}$AS is the best performing algorithm. This is consistent across the 15 out of 18 problem instances where feasibility is found. The non-cp algorithms struggle with feasibility and hence a comparison of the objective values is not meaningful.

The results seen here are similar to those in (Thiruvady et al., 2009) where CP-Beam-$\mathcal{MM}$AS was applied to a single machine job scheduling problem. This is also an optimization problem with non-trivial hard constraints where a complex CP model was devised. Here, with the car sequencing, we similarly see a performance advantage of CP-Beam-$\mathcal{MM}$AS over CP-$\mathcal{MM}$AS. The feasibility differences are also similar with CP-$\mathcal{MM}$AS being slightly more effective. Also like (Thiruvady

et al., 2009), we see that problem specific lower bounds combined with the learning component and CP are not as effective as the stochastic sampling mechanism. This alternative is a simple and generic mechanism to generate estimates and proves to be useful here.

5.3.6 Conclusion

In this study, we have suggested CP-Beam-$\mathcal{MM}$AS for solving the optimization version of the car sequencing problem. Through parallel solution construction, CP-Beam-$\mathcal{MM}$AS is able to effectively optimize the utilization of options across a sequence. This is clearly observable across a number of hard problem instances where CP-Beam-$\mathcal{MM}$AS consistently outperforms CP-$\mathcal{MM}$AS. Furthermore, the feasibility advantages of CP-$\mathcal{MM}$AS are inherited by CP-Beam-$\mathcal{MM}$AS. Similar results were observed during a comparison of the same methods in (Thiruvady et al., 2009).

This result shows that where CP-$\mathcal{MM}$AS may be applied, CP-Beam-$\mathcal{MM}$AS can also be applied with performance gains. This is especially true for complex CP models where constraint propagation is expensive. Furthermore, CP-Beam-$\mathcal{MM}$AS also provides an advantage for those problems where CP is effective but propagation costs are relatively cheap as we have seen with the MMJS problem.

Stochastic sampling proves to be a useful way of obtaining estimates. We explored different lower bounds specific to the car sequencing optimization problem which are not useful when combined with CP. The *uoa* estimate is very effective when large beam widths are usable. However, such a scheme is currently not tractable with respect to memory when combined with CP, due to the large beam width requirements, as shown here.

Chapter 6

Beam Search Estimates

The performance of CP-Beam-ACO is dependent on several factors as we have shown with the previous case studies. One of these is the beam search component which is of vital importance. In particular, the solution construction is guided by the estimates and obtaining these effectively is vital to the progress of the search. In the case studies we used stochastic sampling and we further examine this here. Furthermore, problem specific bounds can also be use to obtain these estimates and we examine a subset of these for each case study.

Regarding stochastic sampling, we have used a relatively straightforward scheme of generating samples based on a relaxed version of the problem and biasing the solution construction with pheromones. This was shown to be very effective and we aim to get insight into this scheme by further analysis of stochastic sampling on the SMJS problem.

A second aim of this chapter is to analyse known bounds for each of the case studies. Typically, bounds focus on relaxed versions of the problem so that solutions are biased towards optimal regions of the search space. We examine such bounds for all three case studies. Bounds for the CS problem have already been discussed in the Section 5.3 of the previous chapter and are summarized here.

6.1 Analysing Stochastic Sampling

One way of determining the effectiveness of stochastic sampling is to examine its usefulness at various levels of the search tree. Consider that we generally use a small number of samples since we need the estimate relatively quickly and cheaply. Intuitively, for instances with a large number of variables, a few samples at the higher levels of the tree are unlikely to provide much guidance since the search space is extremely large. Thus, we hypothesise that while this procedure would be effective at lower levels of the search tree, it is not expected to provide useful estimates higher up in the tree if the number of samples are small. This implies

that for the first few jobs stochastic sampling does not make a significant difference especially for the large instances.

For the first i variables, where stochastic sampling is not used, we compute a lower bound estimate based on the average solution cost of the remaining variables to bind:

$$p + \frac{\mu \times (n - i)}{n} \times \aleph(1.0, 2.0) \tag{6.1}$$

where p is the partial cost so far, μ is the mean setup time estimate, n is the number of variables and $\aleph(1.0, 2.0)$ is a normally distributed random variable with mean 1.0 and variance 2.0. Thus, we add a quantity to the current partial cost that represents an estimate of the cost of the remaining jobs to be sequenced. The normally distributed random variable acts as a smearing factor. This essentially provides an unbiased estimate over the remaining jobs across all partial solutions.

The results for 30 runs per instance and 1000 iterations are shown in Table 6.1. The parameter settings are the same as those used for the SMJS experiments, see Appendix A. The table shows the results for the first $x \div i$ using the estimate in Equation 6.1 while the remaining variables, $x - (x \div i)$, use stochastic sampling. The results here show that increased stochastic sampling always leads to improved feasibility and solution quality. This is despite the small number samples used and we even see an improvement when the first few variables do not use stochastic sampling (see Table 6.2).

These results show that there is consistent improvement with more frequent stochastic sampling. However, as expected, the results with the last three quarters of the variables labeled with stochastic sampling are very close to total stochastic sampling (i.e., all variables). Thus stochastic sampling may be more useful after some proportion of the jobs have been scheduled see the SS column in Table 6.1.

The samples obtained by stochastic sampling are constructed using the pheromone trails. However, if the learning component is eliminated from sampling and the solution components are selected with equal probability, the solution quality can be expected to deteriorate across all runs. These results are shown in Table 6.2 where 30 runs per instance were conducted for 1000 iterations with the same parameters as described in Appendix A. These results show uniform selection performs very well on its own. However, using pheromones still provides an advantage and hence making use of the learning component leads to more effective estimates.

Table 6.1: Beam-ACO using SS; $x \div i$ specifies that the first $x \div i$ variables use equation 6.1 for the lower bound estimate; $x - (x \div i)$ variables use stochastic sampling; Statistically significant results ($p = 0.05$) between the two best performing algorithms for each instance are marked in boldface.

	$x \div 1$				$x \div 2$				$x \div 3$				$x \div 4$			
Instance	best	mean	sd	fail	best	mean	sd	fail	best	mean	sd	fail	best	mean	sd	fail
W8.1	8321	8321.00	0.00	0	8321	8321.00	0.00	0	8321	8321.00	0.00	0	8321	8321.00	0.00	0
W8.2	5818	5823.00	10.26	0	5818	5818.00	0.00	0	5818	5818.00	0.00	0	5818	5818.00	0.00	0
W8.3	4245	4245.00	0.00	0	4245	4245.00	0.00	0	4245	4245.00	0.00	0	4245	4245.00	0.00	0
W20.1	8814	8964.93	89.55	3	8564	8659.67	54.13	0	8504	8563.83	29.20	0	8504	**8543.17**	23.60	0
W20.2	5247	5338.67	38.38	0	5062	5140.50	33.84	0	5062	5099.83	14.12	0	5072	5099.67	12.02	0
W20.3	4577	4680.33	54.65	0	4362	4416.33	33.47	0	4312	4374.00	24.66	0	4312	**4360.33**	19.45	0
W30.1				30	8482	8482.00	0.00	29	8242	8413.14	119.63	8	8117	**8280.67**	92.78	0
W30.2	5262	5450.83	83.52	0	4797	4974.00	69.21	0	4682	4814.00	60.72	0	4627	**4726.67**	45.07	0
W30.3	4588	4853.67	96.43	0	4343	4437.50	61.87	0	4183	4311.33	55.79	0	4183	**4265.17**	38.61	0
RBG10.a	3840	3840.00	0.00	0	3840	3840.00	0.00	0	3840	3840.00	0.00	0	3840	3840.00	0.00	0
RBG16.a	2596	2647.33	51.00	27	2596	2596.00	0.00	0	2596	2596.00	0.00	0	2596	2596.00	0.00	0
RBG16.b	2135	2196.10	35.38	1	2094	2095.63	4.56	0	2094	2094.00	0.00	0	2094	2094.00	0.00	0
RBG21.9	4568	4607.63	19.34	0	4484	4517.33	13.83	0	4481	4498.17	7.67	0	4481	**4492.30**	7.30	0
RBG27.a.3	1166	1346.83	84.72	0	986	1056.20	31.68	0	965	1012.30	23.10	0	957	**1001.03**	24.79	0
RBG27.a.15				30				30	1311	1385.83	56.91	24	1186	**1287.37**	45.65	11
RBG27.a.27				30				30	0			30	1076	1076.00	0.00	26
BRI17.a.3	1039	1086.77	27.08	0	1004	1013.10	6.59	0	1003	1005.83	2.53	0	1003	**1004.87**	2.22	0
BRI17.a.10	1312	1415.40	60.57	25	1035	1076.73	35.68	0	1033	1053.07	15.46	0	1031	**1045.33**	14.96	0
BRI17.a.17				30	1057	1059.27	6.72	8	1057	1057.00	0.00	0	1057	1057.00	0.00	0

Table 6.2: Results using stochastic sampling two different lower bounds; LPE: LP estimate; MSE: minimum setup time estimate; Statistically significant results ($p = 0.05$) between the two best performing algorithms for each instance are marked in boldface.

	SS				SS-UNI				LPE				MSE			
	best	mean	sd	fail	best	mean	sd	fail	best	mean	sd	fail	best	mean	sd	fail
W8.1	8321	8321.00	0.00	0	8321	8321.00	0.00	0	8321	8325.83	0.91	0	8326	8326.00	0.00	0
W8.2	5818	5818.00	0.00	0	5818	5818.00	0.00	0	5863	5894.25	19.32	18	5863	5879.07	22.12	16
W8.3	4245	4245.00	0.00	0	4245	4245.00	0.00	0	4245	4245.00	0.00	0	4245	4245.00	0.00	0
W20.1	8504	**8509.17**	10.30	0	8504	8507.17	7.25	0				30				30
W20.2	5062	**5092.17**	13.36	0	5062	5111.50	20.19	0	5132	5186.00	41.03	0	5112	5177.17	36.80	0
W20.3	4312	**4349.50**	20.50	0	4312	4382.50	25.10	0	4377	4446.33	57.96	0	4377	4473.90	72.14	1
W30.1	7997	**8084.00**	49.11	0	8072	8145.00	52.22	0				30				30
W30.2	4532	**4669.83**	54.02	0	4672	4753.83	45.15	0	4942	5148.00	128.57	25				30
W30.3	4163	**4200.50**	34.76	0	4183	4269.83	35.80	0	4328	4515.50	150.77	4	4328	4421.64	101.62	8
RBG10.a	3840	3840.00	0.00	0	3840	3840.00	0.00	0	3840	3840.00	0.00	0	3840	3840.00	0.00	0
RBG16.a	2596	2596.00	0.00	0	2596	2596.00	0.00	0	2596	2596.00	0.00	0	2596	2596.00	0.00	0
RBG16.b	2094	2094.00	0.00	0	2094	2094.00	0.00	0	2094	2094.00	0.00	0	2094	2094.00	0.00	0
RBG21.9	4481	4485.43	2.84	0	4481	4485.10	3.02	0	7712	7712.00	0.00	0	7712	7712.00	0.00	0
RBG27.a.3	957	**988.83**	17.89	0	984	1040.27	34.21	0	972	988.50	22.28	0	949	992.80	29.27	0
RBG27.a.15	1121	**1202.87**	41.03	0	1262	1318.80	23.55	0				30				30
RBG27.a.27	1076	1076.00	0.00	0	1076	1076.00	0.00	1	1076	1076.00	0.00	18	1076	1076.00	0.00	18
BRI17.a.3	1003	**1005.90**	2.11	0	1003	1008.17	4.24	0	1003	1008.67	3.99	0	1003	1007.90	3.91	0
BRI17.a.10	1031	**1033.33**	3.04	0	1033	1061.87	17.17	0	1031	1031.00	0.00	0	1031	1031.00	0.00	0
BRI17.a.17	1057	1057.00	0.00	0	1057	1057.00	0.00	0	1057	1057.00	0.00	0	1057	1057.00	0.00	0

6.2 Single Machine Job Scheduling

6.2.1 Linear Programming Approximation to the Hungarian Algorithm

The assignment problem can be formulated as a linear program as shown in Chapter 1. The Hungarian algorithm (Kuhn, 2010) can be used to solve this linear program thus providing a lower bound for a partial solution.[1] This is done by considering a relaxed version of SMJS where the setup times of the remaining jobs are considered.

Discussed in Chapter 1 was the assignment problem which is a minimum bipartite matching algorithm (Cormen et al., 2001). It can be formulated similar to the formulation shown in that chapter. We restate it here with minor differences in notation. Given the Graph $G(V, E)$

$$min \sum_{i,j \in V} c_{ij} \times x_{ij} \tag{6.2}$$

subject to

$$\sum_{j \in V} x_{ij} = 1 \qquad i \in V \tag{6.3}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad j \in V \tag{6.4}$$

$$x_{ij} \geq 0 \qquad i, j \in V \tag{6.5}$$

where $x_{ij} = 1$ if the edge between jobs i and j are selected in the final solution. The constraints ensure that there is only one edge selected from any job to all the other jobs. c_{ij} are the setup times between jobs i and j. It can be seen here that we may get cycles which are independent of other cycles in the resulting assignment. Thus we potentially allow selecting setup times between jobs that may not necessarily be consecutive jobs in the final sequence. However, this estimate clearly provides a lower bound to the more constrained version of the problem, SMJS.

6.2.2 Minimum Setup Time

This estimate is obtained by adding the partial cost of the partial solution to the minimum remaining setup times. The latter is computed as follows. For every solution, a complete solution is constructed using the minimum setup time to the next job (j such that $st_{ij} < st_{ik} \forall k$ where i was the last completed job). The lower

[1] Note that the assignment problem is a minimum weighted bipartite matching problem and therefore, flow algorithms cannot be applied here.

bound is:

$$p + \overline{st_{\pi_{ij}}} + \sum_{k=i+1}^{n-1} min_l(\overline{st_{kl}}) \tag{6.6}$$

where $min_l(\cdot)$ is the job such that the setup time from job k to l is minimum.

6.2.3 Comparison

The results of the experiments with the above estimates are presented in Table 6.2. We see here that the LP estimate (LPE) is a good lower bound with respect to makespan but is still outperformed by stochastic sampling. The minimum setup estimate (MSE) performs worse across all instances except the smallest ones that are not tightly constrained (*W8.3,RBG10.a*). With respect to feasibility, LPE and MSE struggle with the tightly constrained instances. There is no real winner when comparing these algorithms. Regarding solution quality, the same can be said, i.e., both implementations perform to a similar level. Furthermore, both implementations are not competitive with the stochastic sampling based implementations with respect to feasibility or solution quality.

6.3 Multiple Machine Job Scheduling

Here we also examine two bounds for the MMJS problem. The first is a quickly obtainable bound based on the current solution weighted-tardiness from which we greedily select the best solutions (See Table 6.3). As these results show the bound does not drive the solutions towards feasibility or improved solution quality. This is expected since we are only considering past experience and not the potential of the remaining components of the solution.

A second more interesting bound - infinite machine infinite resource (IMIR) - is based on a relaxed version of the problem. The first constraint that is relaxed is that jobs that belong to a machine must be placed on the machine. Hence, this bound effectively allows infinite machines. However, precedences between jobs are kept which requires that jobs with precedences must be placed on the same machine. The resource constraint is also relaxed. This is similar to the scheme used by (Stinson, Davis and Khumawala, 1978).

Like stochastic sampling, this bound provides two estimates, the number of violations of the hard due times and the weighted tardiness of the relaxed problem. However, as the results show, the estimate of the violations do not lead the partial solutions towards feasible regions. This result is not surprising since the start times of jobs in the relaxed version of the problem are potentially a lot earlier than their

Table 6.3: Experiments with bounds for the beam search component in CP-Beam-ACS; Tardiness: based on the current partial solution weighted-tardiness; IMIR: infinite machine infinite resource. Statistically significant differences ($p = 0.05$) are shown in boldface.

	Tardiness					IMIR				
	best	mean	sd	fail	iter.	best	mean	sd	fail	iter.
3 - 23	158.08	162.97	4.20	0.00	4.27E3	158.08	**159.99**	1.08	0.00	5.83E3
3 - 53	72.53	72.90	1.81	0.00	5.75E3	72.53	72.62	0.30	0.00	7.39E3
3 - 5	519.06	544.24	27.32	0.00	6.86E3	522.52	550.17	14.47	0.00	5.13E3
4 - 28	24.76	29.38	2.13	0.00	6.99E3	24.10	**28.48**	2.53	0.00	6.35E3
4 - 42	166.50	201.73	23.11	0.00	5.28E3	196.09	287.88	61.91	0.70	9.09E3
4 - 73	98.35	134.61	38.82	0.63	5.02E3	148.32	174.56	22.39	0.80	3.57E3
5 - 21	260.02	**340.28**	48.67	0.00	2.88E3	263.76	342.96	44.30	0.23	3.34E3
5 - 62	553.38	**638.20**	56.86	0.00	3.03E3	527.12	661.39	81.86	0.87	3.23E3
5 - 7				1.00					1.00	
6 - 10				1.00		1455.33	**1640.53**	128.25	0.50	1.17E3
6 - 28	445.36	631.83	104.25	0.10	2.87E3	369.08	**414.23**	28.65	0.00	1.76E3
6 - 58	391.14	478.32	48.95	0.43	2.75E3	430.04	490.12	39.21	0.23	2.65E3
7 - 15				1.00					1.00	
7 - 23	932.67	1076.03	93.45	0.43	2.20E3	780.11	**959.90**	86.43	0.13	1.65E3
7 - 47				1.00					1.00	
8 - 3				1.00					1.00	
8 - 53	638.83	715.18	48.18	0.00	1.54E3	574.90	**644.75**	33.58	0.00	1.33E3
8 - 77	1946.27	**2198.73**	186.70	0.00	1.18E3				1.00	
9 - 20	1519.47	1767.55	147.59	0.00	9.25E2	1348.10	**1541.57**	71.98	0.00	7.03E2
9 - 22	1573.84	1724.21	94.85	0.00	9.71E2	1473.85	1590.51	64.97	0.00	8.08E2
9 - 47	2065.75	2504.39	265.03	0.00	4.95E2	1793.36	**2067.01**	97.34	0.00	5.73E2
10 - 13				1.00					1.00	
10 - 77				1.00					1.00	
10 - 7				1.00					1.00	
11 - 21	1956.16	2578.42	331.02	0.00	6.78E2	1780.09	**1992.25**	121.65	0.00	6.18E2
11 - 56	3306.15	3756.40	326.50	0.70	6.27E2	3053.35	**3477.53**	286.95	0.57	5.07E2
11 - 63				1.00					1.00	
12 - 14				1.00					1.00	
12 - 36				1.00					1.00	
12 - 80				1.00					1.00	

actual start times thereby not violating their hard due time. Therefore, this estimate can be misleading. Considering tardiness, the solution qualities are improved compared to the previous weighted-tardiness estimate but are not comparable to stochastic sampling. Again, here the relaxation of the constraints potentially result in a misleading estimate of the weighted-tardiness leading the solutions into non-optimal areas of the search space.

6.4 Car Sequencing

In Section 5.3 of Chapter 5 we analysed CP-Beam-$\mathcal{MM}$AS with bounds based on the *uoa* and *uua* measures. We briefly review those results here (refer to the tables of results in Section 5.3.3). In terms of feasibility, *uoa*, *uua* and stochastic sampling

perform competitively. The *uoa* estimate is expected to be effective since *uoa* is a cumulative measure of how many option we are over the allowed option in the current sequence. Surprisingly, the *uua* estimate provides results close to that of *uoa*. The combination of the *uoa* and *uua* estimates performs poorly compared ot the other estimates.

In terms of optimizing *uua*, stochastic sampling provides the best solutions where feasible solutions are found. The *uua* bound seems to provide no advantage with the *uoa* bound performing better in several cases.

6.5 Discussion

The estimate for the beam component can be determined by typically relaxing some constraints and obtaining bounds aimed at leading the search towards feasibility/optimality. This allows us to compute principled bounds for the partial solutions and we have examined the following for each problem:

- SMJS: minimum setup time and LP relaxation of the Hungarian algorithm
- MMJS: current tardiness and infinite machine infinite resource
- CS: upper-over assignment and upper-under assignment and their combination

In each case, the respective lower bounds failed to provide the best results. This is observable with respect to both feasibility and optimality. We conjecture here that the reasons for this are twofold. Firstly, consider a number of partial solutions occupying the beam, each of which has an associated lower bound computed. All the implementations of CP-Beam-ACO that we have seen select solutions in a greedy fashion from the partial solutions available. Thus the lower bounds could be misleading and cause better solutions to be discarded from the beam. We have considered introducing some variation into the selection (i.e., probabilistic as opposed to greedy selection) for each problem, however, this did not improve the results. We still believe that further investigation in this direction is likely to lead to useful results.

When considering feasibility, the CP component generally overwhelms any feasibility estimate. Thus, feasibility estimates are not useful in the context of CP-Beam-ACS. However, we see in the CS problem that such an estimate can be effective with Beam-ACO alone. Even considering this improvement, the results are not comparable to the CP implementations. Thus, as far as feasibility goes CP clearly provides the best results.

6.5.1 Stochastic Sampling

Stochastic sampling has been used in a simple form in this thesis. The basic idea is to generate a number of samples using the pheromones. Typically, some of the constraints in the original problem are relaxed and a solution with soft constraints is built in a straightforward manner. The advantages of this method are that it is generic, simple and biased by learning (sampling pheromones). It is generic as it is applicable to a large number of problem classes. We have shown here that it can quite easily be defined for all the problems we have examined. It is simple in that a simplified version of the original problem is considered and a solution can be built in the usual constructive manner of ACO. Finally, it is biased by learning here since we can make sure the samples generated use the pheromones to construct solutions.

The samples usually provide two estimates for the problems considered here. The first is an estimate on feasibility (e.g., in the SMJS problem: for a given sequence, the number of jobs that violate their due times) and the second is the solution cost (e.g., in the SMJS problem: makespan of sequence). We have seen that the feasibility estimate is useful for SMJS (when CP is not used, i.e., Beam-ACO is more effective than ACO) and MMJS problems. With the MMJS problem, the CP component is relatively weak, and here we see a big gain from stochastic sampling. However, it should be noted that this gain is mainly useful when combined with CP. In general, as we have seen with the lower bounds, the feasibility advantage provided by stochastic sampling is not significant for complex CP models (SMJS and CS problems).

The optimality estimate proves to be useful for the SMJS and CS problems. This is not the case for the MMJS problem. The former two problems have relatively complex CP models and the MMJS problem has a simpler CP model. Despite this, the more likely reason that this estimate was unsuccessful with MMJS is due to the following. Several permutations may map to the same point[2] on the fitness landscape and a small change in the permutation could result in a very different position on the landscape. Therefore, the samples may more often than not, mislead the solution construction.

We have also analysed the contribution of stochastic sampling at various levels of the search tree. This was done with the SMJS problem. The results showed that the samples are largely useful after a quarter of the variables have been labeled. From this point onwards there was always a significant reduction in performance, i.e., sampling after 1/4 variables >> 1/2 variables >> 3/4 variables. This was despite the relatively small number of samples compared to solution space (e.g. 10

[2]Note that this is not the same objective function value but the same point.

samples for a 30 variable problem). Overall, stochastic sampling has been effective, more effective than the lower bounds, but why it works so well is still surprising.

6.6 Conclusion

In this chapter we analysed the estimates of the beam search component of the algorithms developed in this thesis. We revisited the SMJS problem to determine the effectiveness of stochastic sampling. We find that relatively few samples provide a positive effect after a quarter of the variables have been labeled consistent with (López-Ibáñez et al., 2009). For each case study we examined problem specific bounds. These bounds showed varying results, however, the final outcome is that stochastic sampling was more effective than any particular bound across all problems.

Stochastic sampling provides a way to obtain estimate relatively cheaply in a problem independent fashion. Compared to other estimates/bounds, this estimate provides the additional estimate of the feasibility of a path. This is particularly useful in the context of the problems studies in this thesis where problems are tightly constrained. Having gained some insight and observed its effectiveness, the success of this technique still requires further investigation. This characteristic can be viewed as being closely linked with the use of metaheuristics, where they are particularly useful if there is not principled approach to solving the problem.

The problem specific bounds were also analysed but mostly shown to be inferior particularity in terms of feasibility. While the CP component leads solutions towards feasible regions the greedy selection based on a bound is ineffective. There is certainly room to understand this more thoroughly but replacing greedy selection by introducing variability dependent the partial solution quality. Furthermore, the selection based on the estimate could be swapped with the pheromone selection. This introduces variability at both stages of candidate selection potentially leading to improved solution costs.

Chapter 7

Conclusions and Future Work

This thesis shows that real-world tightly constrained COPs can be tackled by efficiently hybridizing CP with ACO. We see here that the proposed algorithm, CP-Beam-ACO, is a viable implementation of CP-ACO in practical settings. CP-ACO was shown to be effective by Meyer & Ernst (2004) on the SMJS problem. However, this algorithm had run-time problems due to the overhead of constraint propagation. Thus, obtaining high quality solutions for large problem instances in a reasonable time-frame was not possible. This motivated the need to improve CP-ACO in practical settings and we have achieved this by incorporating the solution construction mechanism of Beam search in ACO. This leads to significant performance gains with the resulting hybrid being more efficient and effective.

The main issue with CP-ACO was that the ACO component of the algorithm frequently reconstructs (parts of) a solution requiring repetition of the same propagation steps. In CP-Beam-ACO we overcome this problem by replacing the standard ACO component of CP-ACO with Beam-ACO (Blum, 2005). The underlying idea behind this hybrid was to build unique solutions and we achieved this effectively via Beam-ACO. Like ACOs solution construction, Beam search builds solutions in parallel. However, the solutions are partially dependent on each other in two aspects: (1) always requiring that a partial solution be extended by a uniquely new component and (2) partial solutions may be discarded in favour more promising partial solutions in another part of the search tree. Hence, within an iteration, the same propagation is never repeated and the algorithm biases the search towards more optimal regions. By using an efficient CP solver (Gecode) we showed CP-Beam-ACO provides significant improvements over CP-ACO.

In order to demonstrate the effectiveness of CP-Beam-ACO, we examined its performance on three tightly constrained COPs. These were the SMJS, MMJS and CS problems which are examples of typically occurring problems in real-world settings. Often the unconstrained version of the problem is itself $\mathcal{NP}$-hard making it complex to solve. When constraints are introduced, these problems can become

significantly harder to solve. Through CP-ACO, feasible solutions to the problems are obtained but with the overhead of large run-time requirements. By parallelizing the solution construction we showed that CP-Beam-ACO outperforms CP-ACO retaining its advantages. The metaheuristics, ACO and Beam-ACO, struggle to find feasible solutions and perform significantly worse than either CP-based algorithm. Thus, where strong CP models can be defined in conjunction with effective learning models we see a clear advantage of using CP-Beam-ACO.

In terms of solution quality we see that CP-Beam-ACO significantly outperforms CP-ACO for problems where complex CP models have been defined. We see for the SMJS and CS problems that CP-Beam-ACO is almost always superior irrespective of problem size. This can be attributed to two factors. The first is that CP-Beam-ACO spends less time in CP propagation or the feasibility component of the algorithm. This saved time can be spent to locate high quality solutions where feasibility is found. This is observed with the larger number of nodes labelled as seen with CS. Secondly, the beam structure is more suited to intensification by biasing solutions towards optimal regions via the estimates. This is clearly seen in the case of SMJS where Beam-ACO is also superior to ACO. This comparison is not so straightforward with CS since both ACO and Beam-ACO do not find a single feasible solution.

CP-Beam-ACO inherits the feasibility advantage of CP-ACO. This can be seen with all three problems we have examined. Where complex CP models have been implemented we see that CP-ACO has a slight feasibility advantage. This is possibly attributable to a combination of two factors. The first is that CP-ACO spends more time in propagation as it does not require the additional overhead of CP-Beam-ACO such as the estimates. More importantly, the beam search estimate biases the solution construction towards optimal regions thereby reducing variation in the current solutions being searched. If there are relatively few feasible regions, these may not be explored as the estimate may eliminate them early on in the search. Having pointed this out, the feasibility advantage of CP-ACO over CP-Beam-ACO across both problems is less than 1%.

CP-Beam-ACO has a significant feasibility advantage for the MMJS problem. We have devised a relatively simple CP model for this problem and the time spent in propagation is very little even with increasing problem size. Thus the additional time needed by CP-Beam-ACO to compute the estimate is much larger than CP propagation, but crucially, the estimates guides the solutions to feasible regions. We see a 22% improvement across the runs conducted with the MMJS problem instances with CP-Beam-ACO over CP-ACO.

A comparison of feasibility of the CP-based and non-CP algorithms shows that the former class of methods struggle to find feasibility where complex CP models can be defined for a problem. This is clear with the CS problem but is also seen

with the SMJS problem for large instances. This is not surprising given that the CP component is specially designed to identify feasibility and more time spent in this component should lead to feasibility. ACS for the MMJS problem performs nearly as well as CP-ACS. Thus, the relatively simple CP model leads to small gains. However, in conjunction with stochastic sampling and the beam structure, the CP model proves very effective leading to significantly improved feasibility overall.

The estimates driving the beam component of CP-Beam-ACS have been analysed for all three problems. We suggest stochastic sampling (López-Ibáñez and Blum, 2008) as a generic method to obtain estimate cheaply. This technique is shown to be effective on all the problems tackled. It serves as an alternative to principled bounds which may not be obtained cheaply. We see that in the context of solution quality, however, a feasibility advantage may also be obtained. Overall, in our case studies stochastic sampling is more effective than problem specific bounds.

For each study considered we examined known bounds from the literature to determine if they would provide an advantage over CP-ACO. However, this was not the case for any of the bounds across all the studies. The basic problem identified here is that, due to the greedy selection based on the estimates, promising feasible solutions are often discarded to favour higher quality solutions. Thus, at higher levels of the search tree where infeasibility cannot be determined, future infeasible solutions which are locally promising may be selected. A potential solution to this is to introduce variability in the selection. Furthermore, estimates based on feasibility in conjunction with the bounds may also be useful. These two suggestions are left to be addressed in the future.

7.1 Limitations of CP-Beam-ACO

The main component where CP-Beam-ACO differs from CP-ACO is with Beam search. In this context, cheaply computable estimates are crucial for the success of CP-Beam-ACO. We have shown through three case studies that stochastic sampling is a simple way of obtaining quick estimates by relaxing some constraints. However, we also see that problem specific bounds are not as useful. While one explanation could be due to the lack of variation in the selection based on the estimates, the main problem seen is the lack of feasible solutions. Thus, through the estimates, the beam structure reduces the number of feasible regions favouring optimal ones thereby undoing the work of CP and this aspect is worth examining further.

We have shown that CP-Beam-ACO provides a significant advantage where CP-ACO is usually an effective algorithm. One set of limitations of CP-Beam-ACO can therefore be directly mapped to issues that CP-ACO might have on a particular

problem. We now discuss two different problems where CP-ACO fails and thus CP-Beam-ACO is ineffective.

The traveling tournament problem (see Appendix E) is a good example of where a reasonable pheromone model is hard to determine. The problem is the double round robin tournament where we must determine a schedule for n teams with $2 \times (n-1)$ rounds. Every team must play every other team, home and away, subject to home and away restrictions.

This problem structure requires a learning scheme to take into account dependencies between home and away locations of a team at the previous and current rounds. It was shown by (Crauwels and Oudheusden, 2003) that a simplistic scheme can derived from an $O(n^2)$ pheromone matrix where the current team being selected is dependent on the previous team is insufficient. We further tested more complex pheromone learning matrices without successful results (see Appendix E).

The basic problem here is that there are far too many parameters to learn with increasing problem size. Even though CP approaches for this problem have shown to be effective in combination with other methods (Benoist, Laburthe and Rottembourg, 2001; Easton, Nemhauser and Trick, 2003), an ineffective ACO component renders CP-ACO ineffective leading to poor performance by CP-Beam-ACO.

The photo album layout problem was discussed in Section 2.2 of Chapter 2. Briefly, the problem consists of placing a number of photos on a page such that the relative photo areas are specified to be as close to the desired photo areas as possible. The solution to the problem is specified via a number of constraints between photos, i.e., how any two photos should be placed relative to each other. Beam search was shown to be effective for this problem and we further showed that we could add a learning component resulting in Beam-ACO which is also effective.

However, despite the constraints involved in the problem, the CP model proves ineffective here. The reason for this is that photos are incrementally added to the layout and their sizes change. A larger number means that the photos are smaller on the page even though their relative sizes are maintained. Therefore, the domains associated with the variables change when photos are added to the layout and previously feasible solutions may not be feasible any more. Such a scheme does not allow the CP model to be useful, hence, CP-ACO is ineffective for this problem.

7.2 Further Improvements

There is certainly room for improvement regarding CP-Beam-ACO. We have seen that by parallelizing the solution construction of CP-ACO we have significant improvements by avoiding re-propagation. However, given the iterative nature of ACO,

the same problem persists across iterations. Therefore, by saving solver states from iteration to iteration, we can expect further improvements to CP-Beam-ACO.

One aspect of the parallel solution construction that leads to an inefficiency is that solver states for each solution in the beam must be maintained. The memory requirements for the combination of solutions is extremely large and could exponentially increase with problem size. This limits how large a beam width we may use. As we see with the CS problem, this limitation potentially limits designing more effective solutions. Thus, efficiently managing memory is worth examining.

An extension of the work done here would be to design a way to systematically assess where CP-Beam-ACO is likely to be successful. For different problem types, if a mechanism can be designed to automatically suggest if CP-ACO and CP-Beam-ACO are likely to be successful and further quantify this success, then such a system will be very useful to practitioners designing algorithms for similar types of problems.

CP-ACO itself can be much improved by considering a parallel framework. This is particularly interesting given the availability of resources (e.g., clusters). Consider a master-slave framework. Here, the ACO algorithm would be the master which could construct each solution independently on separate machines (slaves). While the performance of this algorithm is dependent on "slowest slave," the gain in time improvements are likely to be significant.

This leads to a parallel version of CP-Beam-ACO. Since this algorithm parallelizes CP-ACOs solution construction, extending it to a parallel setting appears promising. There are further aspects to consider compared to CP-ACO. Firstly, the solutions are not independent any more. Thus communication between the solutions are crucial. In particular, to make the parallel version efficient, the solver states must by located on shared memory so that solutions may share solver states. While this can be done effectively, the limit on the beam widths are still a potential problem. Considering all aspects, it is certainly worth examining parallel version of CP-ACO and CP-Beam-ACO. The parallel component could permit applying the algorithm to much larger problems with many more variables.

7.3 Summary

We see from the studies in this thesis that CP-Beam-ACO is an effective algorithm for COPs with non-trivial hard constraints. Problems with these characteristics for which ACO and CP models can be effectively defined can be solved by CP-ACO. From the studies conducted here, we see that where CP-ACO may be applied CP-Beam-ACO may also be applied with gains by eliminating the inefficiency of ACOs repeated solution construction. This leads to an efficient implementation. Thus,

to solve this class of problems in practical settings, CP-Beam-ACO is the ideal algorithm.

We have already seen significant improvements to CP-ACO. Furthermore, we have suggested ways in which CP-Beam-ACO might be improved. The possibilities are mainly in two directions. The first is to consider repeated solutions across iterations and keeping track of repeating solver states. This will lead to improvements by further reducing repeated propagation. The second is to extend CP-Beam-ACO into a parallel setting. Given the run-time requirements of CP propagation this is promising direction to explore.

A further possibility to consider is the integration of CP and other metaheuristics. We have seen with ACO that the integration is straightforward and other studies have considered hybrids with Genetic algorithms (Barnier and Brisset, 1998). However, other hybrids with BOA (Pelikan, Goldberg and Cantú-Paz, 1999) for instance are potentially interesting. Here, the CP model and bayesian network could share information to more effectively explore search spaces. For example, high probability values for variables could be used to post constraints to the CP solver where these variables are labeled with the value. On the other hand, inferred constraints may be used to update probabilities in the bayesian network.

Another possible area of future work is to explore other hybrids, for example, CP and mixed integer programming or metaheuristics with mixed integer programming. Some of these have already been discussed earlier with studies demonstrating their potential. Each class of methods are effective in certain settings and their relative advantages may be used by hybrids to deal with different problem types.

In conclusion, this thesis has demonstrated that CP-Beam-ACO is an efficient way to implement constraint-based ACO and provides a suitable framework to tackle COPs with non-trivial hard constraints. The algorithm's effectiveness on three case studies is clearly seen thus making it a promising option to solve other such problems in the future.

Appendix A

CP-Beam-ACO Settings

Table A.1 shows the configuration details of CP-Beam-ACO for each of the three problems.

Table A.1: CP-Beam-ACO run configurations.

	runs	exe. time	# instances	size (#variables)	restarts
SMJS	30	10 hrs	43	8 - 192	✓
MMJS	30	1 hr	30	10 - 130	✓
CS	30	15 hrs	18	100 - 500	✗

Table A.2 shows the parameter settings for ACO and the Beam component of the algorithms.

Table A.2: CP-Beam-ACO parameter settings for the three case studies.

	ACO								Beam		
	variant	n_a	ρ	α	β	q_0	τ_{min}	τ_{max}	θ	μ	N^s
SMJS	$\mathcal{MM}$AS (hypercube)	10	0.01	1.0	1.0	0.5	0.001	0.999	10	2.0	20
MMJS	ACS	10	0.1	1.0	1.0	0.5	-	-	10	3.0	5
CS	$\mathcal{MM}$AS	10	0.02	1.0	6.0	0.5	0.01	4.0	10	3.0	10

Appendix B

Strip Packing Instances and Results

The next section gives details of the instances and provides some details the experimental setup and results. For the complete results see (Thiruvady et al., 2008).

B.1 Instances and Results

Table B.1 provides details of the instances used in (Thiruvady et al., 2008) whcih come from (Hopper and Turton, 2001). There are seven categories of problems with each category consisting of approximately the same number of items to be placed on the strip (e.g., Category 3 has 3 instances with either 28 or 29 items). Each category has a known optimal height (a dense packing) and a pre-defined width which are also specified in the table.

Table B.1: Problem Instances for Strip Packing.

Problem	C1	C2	C3	C4	C5	C6	C7
Size	16-17	25	28-29	49	72-73	97	196-197
Height	20	15	30	60	90	120	240
Width	20	40	60	60	60	80	160

The following table (Table B.2) provides all the results for the algorithms for each category of the SPP problem. The table presents best results, %-gap to theoretical optimum, average and standard deviation over 100 runs. The columns represent each category of problems (C1-C7), the values for each category are averages over the three problem instances in the category.[1] Best results for each category (if unique) are marked in bold face where the differences are statistically significant.[2]

[1] Recall that the instances of each category have the same optimum objective value.

[2] No best results are marked where all algorithms find the global optimum.

The percentage values indicated are the gaps to the theoretical optimum, and are calculated as $\frac{Best-Optimum}{Optimum}$, similarly for the mean.

Complete results for all the dataset are shown in Table B.3 where the order in not learnt and Table B.4 shows those results where the order was learnt. The column *iter* in Table B.4 specifies the average number of iterations when the best soltion was found. Tests for statistical significance for the comparison of results were conducted and these are shown in Table B.5.

B.2 Examples of Packings on Selected Instances

Here we show an example on a run on one of the largest instances. Given a random order, BLF itself is able to generate very good placements on the strip (see Figure B.1(a)). However, when the placement order is learned we can see large improvement visually with smaller gaps in the placement.

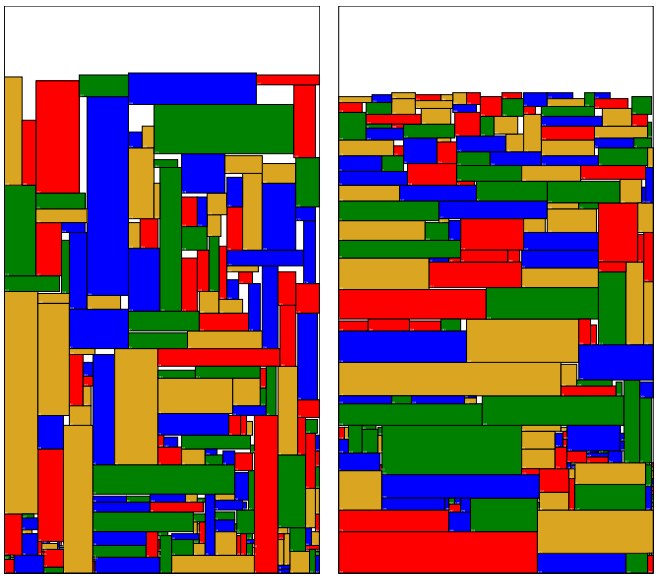


Figure B.1: Results from a problem instance with 197 items and was run for 500o iterations; (a) A packing obtained with using a uniformly random order (b) A packing obtained with order learning and using both placements.

Table B.2: Summary of Results by Category. Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

Problem	C1	C2	C3	C4	C5	C6	C7
				RR			
Best	20	16	31	62	93	126	251
% best	0	7	3	3	3	5	5
μ	20.9	16.0	32.2	63.9	95.1	127.3	254.1
$\%\mu$	4	7	7	7	6	6	6
σ	0.24	0.00	0.23	0.36	0.51	0.48	0.69
				RRF			
Best	20	15	31	62	92	124	248
% best	0	0	3	3	2	3	3
μ	20.7	16.0	31.4	62.7	93.7	125.3	251.0
$\%\mu$	3	6	5	5	4	4	5
σ	0.27	0.12	0.22	0.36	0.36	0.43	0.52
				LR			
Best	20	15	**30**	62	93	126	251
% best	0	0	**0**	3	3	5	5
μ	**20.0**	15.8	31.3	63.8	95.0	128.1	254.1
$\%\mu$	**0**	5	4	6	6	7	6
σ	0.00	0.33	0.42	0.46	0.46	0.58	0.77
				*ACS*1			
Best	20	16	31	61	92	123	245
% best	0	7	3	2	2	2	2
μ	20.8	16.0	31.4	62.2	92.9	123.6	247.9
$\%\mu$	4	7	5	4	3	3	3
σ	0.31	0.00	0.48	0.34	0.24	0.23	0.47
				*ACS*2			
Best	20	16	31	61	92	122	245
% best	0	7	3	2	2	2	2
μ	20.8	16.0	31.5	62.5	93.0	123.8	248.4
$\%\mu$	4	7	5	4	3	3	3
σ	0.35	0.00	0.46	0.33	0.26	0.40	0.51
				*ACS*3			
Best	20	15	31	61	92	122	245
% best	0	0	3	2	2	2	2
μ	20.8	16.0	31.4	62.2	92.9	123.6	247.9
$\%\mu$	4	7	5	4	3	3	3
σ	0.36	0.07	0.47	0.33	0.26	0.30	0.45
				*ACS*4			
Best	20	15	31	61	**91**	122	**243**
% best	0	0	3	2	**1**	2	**1**
μ	20.4	15.6	**31.0**	**61.6**	**92.0**	**122.7**	**246.3**
$\%\mu$	2	4	**3**	**3**	**2**	**2**	3
σ	0.43	0.36	0.07	0.25	0.28	0.08	0.48

Table B.3: Complete Breakdown of Results Without Learning Order. Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

	RR				*RRF*				*LR*			
	Best	μ	σ	Iter.	Best	μ	σ	Iter.	Best	μ	σ	Iter.
C1-1	20	20.9	0.3	202.6	20	20.2	0.4	1260.0	20	**20.0**	0.0	65.7
C1-2	21	21.0	0.0	640.6	21	21.0	0.0	46.0	**20**	**20.0**	0.0	1266.5
C1-3	20	20.8	0.4	554.7	20	20.8	0.4	648.3	20	**20.0**	0.0	52.1
C2-1	16	16.0	0.0	688.6	15	16.0	0.2	124.1	15	**15.8**	0.3	416.1
C2-2	16	16.0	0.0	329.6	16	16.0	0.0	4.9	**15**	15.9	0.2	181.5
C2-3	16	16.0	0.0	285.2	15	16.0	0.2	54.4	15	**15.5**	0.5	1295.9
C3-1	31	32.0	0.1	650.7	31	31.0	0.0	693.2	**30**	31.0	0.3	1204.8
C3-2	32	32.5	0.5	1344.9	31	**31.9**	0.2	313.3	**30**	32.0	0.2	976.7
C3-3	31	32.0	0.1	917.8	31	31.3	0.5	1485.1	**30**	**31.0**	0.7	1649.4
C4-1	63	64.1	0.3	1487.5	**62**	**62.9**	0.2	908.7	63	64.2	0.5	1721.4
C4-2	63	63.9	0.3	1206.4	**62**	**62.9**	0.3	695.1	63	64.1	0.4	1420.3
C4-3	62	63.8	0.4	1067.7	62	**62.4**	0.5	1480.6	62	63.1	0.5	1889.6
C5-1	93	94.3	0.5	1439.4	93	**93.1**	0.3	1647.9	93	94.0	0.3	1303.5
C5-2	94	95.7	0.5	1764.5	**92**	**93.9**	0.3	1110.3	94	95.8	0.6	1849.5
C5-3	94	95.3	0.5	1581.2	**93**	**94.1**	0.4	1202.4	94	95.0	0.4	1852.0
C6-1	127	127.9	0.5	1657.4	**124**	**125.8**	0.5	1205.3	127	128.5	0.6	1498.6
C6-2	126	127.3	0.5	1711.3	**124**	**125.0**	0.4	1448.6	127	128.6	0.6	1692.0
C6-3	126	126.8	0.4	1631.5	**124**	**125.2**	0.4	1302.5	126	127.2	0.5	1633.8
C7-1	253	254.7	0.6	1776.9	**251**	**252.5**	0.6	1612.4	253	254.6	0.6	1617.1
C7-2	251	253.3	0.8	1648.3	**248**	**249.3**	0.5	1675.9	251	254.1	1.0	1984.5
C7-3	253	254.2	0.7	1852.6	**250**	**251.1**	0.5	1771.1	251	253.8	0.8	1930.0

Table B.4: Complete Breakdown of Results with Learning Order. Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

	$ACS1$				$ACS2$				$ACS3$				$ACS4$			
	Best	μ	σ	Iter.	Best	μ	σ	Iter.	Best	μ	σ	Iter.	Best	μ	σ	Iter.
C1-1	20	20.8	0.4	458.0	20	20.8	0.4	715.1	20	20.8	0.4	563.2	20	**20.2**	0.4	1107
C1-2	21	21.0	0.0	30.2	20	21.0	0.1	96.0	20	21.0	0.2	98.3	20	20.8	0.4	347.7
C1-3	20	20.5	0.5	847.1	20	20.6	0.5	905.0	20	20.6	0.5	726.7	20	20.3	0.4	1219.5
C2-1	16	16.0	0.0	48.6	16	16.0	0.0	133.1	16	16.0	0.0	44.9	**15**	15.9	0.2	198.1
C2-2	16	16.0	0.0	21.8	16	16.0	0.0	44.3	15	16.0	0.1	59.4	15	**15.7**	0.4	634
C2-3	16	16.0	0.0	8.8	16	16.0	0.0	13.3	15	16.0	0.1	32.1	15	**15.2**	0.4	1342.3
C3-1	31	31.6	0.5	810.7	31	31.7	0.5	839.3	31	31.6	0.5	944.9	31	**31.0**	0.0	100.4
C3-2	31	31.5	0.5	1178.0	31	31.6	0.5	1151.9	31	31.4	0.5	1496	31	**31.0**	0.2	337
C3-3	31	31.3	0.5	1366.6	31	31.3	0.5	1305.8	31	31.2	0.4	1465.7	31	**31.0**	0.0	87.7
C4-1	62	62.7	0.5	1015.7	62	63.0	0.3	911.1	62	62.8	0.4	951.3	**61**	**62.0**	0.2	330.8
C4-2	61	62.0	0.3	1605.9	62	62.5	0.5	1192.8	61	62.1	0.3	1383.1	61	**61.7**	0.5	887.8
C4-3	61	61.9	0.3	505.5	61	61.9	0.2	408.7	61	61.9	0.2	511.6	61	**61.0**	0.1	655.8
C5-1	92	93.0	0.1	842.7	92	93.0	0.2	1135.5	92	92.9	0.2	879.2	**91**	**92.0**	0.1	1154.7
C5-2	92	92.8	0.4	1241.1	92	93.0	0.1	1394.3	92	92.8	0.4	1204.7	**91**	**91.8**	0.4	663.1
C5-3	92	93.0	0.2	1466.4	92	93.2	0.4	1667.5	92	93.0	0.2	1589.7	92	**92.1**	0.3	1554
C6-1	123	123.9	0.3	1115.6	123	124.1	0.4	1525.8	123	123.8	0.4	1106.4	**122**	**123.0**	0.1	492.3
C6-2	123	123.0	0.1	1461.2	122	123.3	0.5	1519.6	122	123.0	0.2	1316.6	122	**122.0**	0.0	730.8
C6-3	123	124.0	0.3	1217.7	123	124.1	0.4	1528.1	123	123.9	0.3	1140	123	**123.0**	0.1	1083.2
C7-1	248	249.7	0.6	1851.6	249	250.2	0.5	1811.1	249	249.7	0.5	2001.5	**247**	**248.3**	0.5	1807.2
C7-2	245	245.9	0.3	1991.3	245	246.3	0.5	1723.0	245	245.9	0.3	1764.1	**243**	**244.3**	0.5	1702.4
C7-3	247	248.3	0.5	2173.9	248	248.7	0.5	1756.1	247	248.2	0.6	2334.1	**245**	**246.4**	0.5	1850.1

Table B.5: p-values of a two-tailed t-test with unequal variances for pairwise comparisons between selected algorithms; Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

	RR vs RRF	RR vs LR	RRF vs LR	$ACS1$ vs $ACS2$	$ACS1$ vs $ACS3$	$ACS3$ vs $ACS4$	RR vs $ACS4$	RRF vs $ACS4$	LR vs $ACS4$
C1-1	**1.60e-05**	**7.80e-18**	1.20e-01	1.00e+00	1.00e+00	**9.50e-04**	**1.6e-05**	1.0e+00	1.2e-01
C1-2	1.00e+00	**0.00e+00**	**0.00e+00**	1.00e+00	1.00e+00	1.60e-01	1.2e-01	1.2e-01	**1.7e-09**
C1-3	1.00e+00	**1.70e-09**	**1.70e-09**	6.60e-01	6.60e-01	1.40e-01	**5.7e-03**	**5.7e-03**	**1.9e-02**
C2-1	1.00e+00	**3.60e-02**	8.10e-02	1.00e+00	1.00e+00	1.20e-01	1.2e-01	2.6e-01	3.8e-01
C2-2	1.00e+00	1.20e-01	1.20e-01	1.00e+00	1.00e+00	**2.20e-02**	**1.9e-02**	**1.9e-02**	1.6e-01
C2-3	1.00e+00	**1.80e-03**	**3.70e-03**	1.00e+00	1.00e+00	**4.50e-09**	**1.7e-09**	**5.3e-08**	1.4e-01
C3-1	**2.50e-79**	**2.60e-19**	1.00e+00	6.60e-01	1.00e+00	**2.00e-04**	**2.5e-79**	1.0e+00	1.0e+00
C3-2	**5.30e-04**	**3.70e-03**	2.60e-01	6.60e-01	6.60e-01	**2.00e-02**	**6.4e-16**	**1.7e-19**	**8.2e-23**
C3-3	**2.30e-05**	**1.30e-05**	2.70e-01	1.00e+00	6.20e-01	1.20e-01	**2.5e-79**	5.9e-02	1.0e+00
C4-1	**7.40e-21**	5.90e-01	**9.40e-13**	1.10e-01	6.20e-01	**5.30e-08**	**8.6e-45**	**1.7e-19**	**4.3e-28**
C4-2	**2.80e-12**	2.10e-01	**1.20e-12**	**7.30e-03**	4.60e-01	**3.10e-02**	**4.4e-25**	**6.1e-10**	**7.7e-25**
C4-3	**6.30e-11**	**6.70e-04**	**2.00e-03**	1.00e+00	1.00e+00	**1.70e-27**	**1.3e-53**	**1.4e-15**	**2.1e-28**
C5-1	**6.10e-10**	1.10e-01	**2.00e-10**	1.00e+00	1.60e-01	**1.70e-27**	**3.2e-32**	**2.8e-22**	**2.1e-49**
C5-2	**1.30e-18**	6.90e-01	**2.50e-16**	1.30e-01	1.00e+00	**7.50e-08**	**2.9e-47**	**3.3e-29**	**3.5e-42**
C5-3	**1.40e-08**	1.40e-01	**1.10e-06**	1.60e-01	1.00e+00	**2.00e-13**	**1.3e-41**	**2.9e-27**	**1.5e-44**
C6-1	**1.50e-17**	**1.60e-02**	**4.50e-22**	2.10e-01	5.30e-01	**4.50e-09**	**1.7e-76**	**1.2e-41**	**3.2e-72**
C6-2	**2.40e-23**	**3.70e-07**	**7.00e-37**	6.40e-02	1.00e+00	**5.90e-37**	**1.5e-83**	**9.1e-60**	**2.8e-86**
C6-3	**2.70e-16**	**5.00e-02**	**5.80e-19**	5.30e-01	4.60e-01	**1.90e-16**	**1.5e-73**	**3.6e-40**	**7.0e-66**
C7-1	**3.00e-14**	7.10e-01	**3.00e-13**	**4.40e-02**	1.00e+00	**2.30e-09**	**1.6e-65**	**1.4e-40**	**1.7e-64**
C7-2	**1.40e-29**	**5.00e-02**	**4.20e-30**	**3.10e-02**	1.00e+00	**1.50e-15**	**5.6e-76**	**4.5e-56**	**4.4e-70**
C7-3	**1.80e-23**	2.40e-01	**1.30e-16**	7.50e-02	6.90e-01	**7.30e-12**	**2.0e-72**	**2.5e-52**	**1.2e-62**

Appendix C

Photo Album Layout

C.1 BRIC

A brief description of the BRIC algorithm is provided here. A layout is constructed by considering a number of photos specified by a binary tree. The internal nodes of the tree are always H or V (horizontal or vertical, respectively) and the leaves are photos. The internal nodes specify how a particular photo will be placed with respect to another, i.e., either horizontally to the left or right or vertically to the top or bottom of the other photo. See Figure C.1. This example shows how a binary tree may be used to represent a layout of photos. All the internal nodes represent cuts (how the photos are required to be split) which can either be horizontal or vertical and the leaves represent photos. Here, we can also see how a layout is constructed from the tree representation.

For each node in the tree, a *bounding box* is computed. This can be seen for example with the node H_9. The purpose of this box is to specify that all the photos to be placed within the box must be have appropriate widths and height such that they fit in it. This is enforced as follows. The layout is obtained by solving $n-1$ linear equations for the $n-1$ inner nodes such that the heights/widths of all photos to be separated by vertical/horizontal cuts must sum to the height/width of the bounding box. An n^{th} equation is obtained by constraining either the hight/width of the root bounding box to the height/width of the canvas which ensures that one of the canvas dimensions is met. As a result, two separate sets of equations can be solved constraining either the heights or widths. For one of these all the photos will fit on the canvas and this is chosen as the solution to proceed with. For further details of the construction of the layout see (Atkins, 2008).

The binary tree itself is constructed in the following manner. The complete layout is constructed from scratch incrementally. An initial tree, T_1 consists of a single photo that takes up the whole canvas. To this a second photo is added, a score

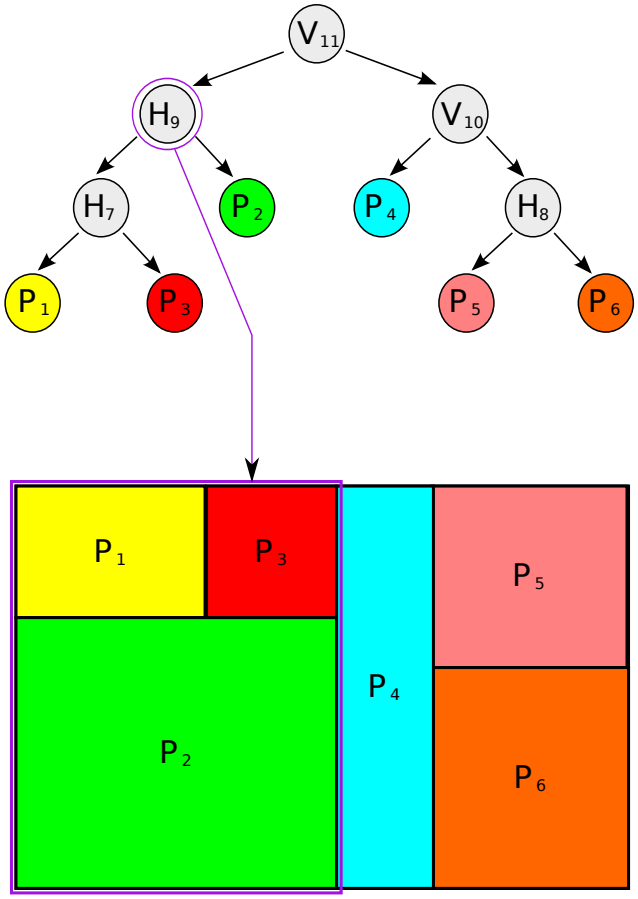


Figure C.1: This figure shows how a layout can be represented by a binary tree. All the internal nodes are either H or V representing horizontal or vertical cuts, respectively. This implies that all nodes at the lower level of the tree will be split with a horizontal or vertical line. For example, the node H_9 specifies that its left and right children must be split by a horizontal cut. The leave represent photos, e.g. P_5. The result is that P_2's with is the combined with of P_1 and P_3. Also highlighted, is the bounding box around H_9. The resultant mapping is shown on the canvas.

computed, and this process continues until T_n is obtained where all n photos have been placed on the layout. For more than two nodes, a location for placing the new photo needs to be selected. This is done by exhaustively attempting every possible node to split on (including trying horizontal and vertical cuts) and then selecting the m best solutions (sorted by score) to proceed with. Some of these solutions may not permit a feasible solution either where some computed heights/widths may negative. Such solutions are discarded.

C.2 Order Learning

The order of the photos can be obtained in three different ways. The first is to fix an order for the whole duration of the search. This has no effect on the BRIC

algorithm which will deterministically create the same solution of each iteration. However, if we include a learning component, the solution constructed will converge to a solution only when the pheromones have converged. The second method is to fix an order for a single iteration (uniformly randomly across all photos) and allow this order to change from iteration to iteration. This potentially has an effect on BRIC and BRIC-ACO. The third and final method randomly selects a picture to be added to the partial solution. This has the effect that at any time several solutions with variable orders may be placed in the beam. This method also affects both algorithms. The BRIC algorithm tested here uses random orders during the solution construction.

For the second and third algorithms, the pheromone model is to pick a photo given the previous photo in the sequence:

$$\mathbf{p}(\pi_i = j) = \frac{\tau_{\pi_{i-1}j}}{\sum_{d \in D \setminus \{\pi_1, \ldots, \pi_{i-1}\}} \tau_{\pi_{i-1}d}} \tag{C.1}$$

A further pheromone model is used in addtion to the above model[1]. For each photo, two pheromone models for horizontal and vertical cuts with every other photo. For a new photo j at position i, a joint probability is computed for the partial solution as follows:

$$\mathbf{p}(\hat{\pi}) = \prod_{k=1}^{i-1} \hat{\tau}_{\pi_k, j, c(\pi_k, j)} \tag{C.2}$$

where $c(\pi_k, j)$ is the cut of photo j associated with photo π_k. $\hat{\tau}_{\pi_k, j, c(\pi_k, j)}$ is the pheromone associated with these two photos given their cut.

At the end of each iteration, three different solutions are potentially used to update the pheromones. See (Thiruvady et al., 2009) for these details, but briefly, for each pair of photos i, j and a solution π:

$$\tau_{ij} \leftarrow min(max(\tau_{ij} \times (1.0 - \rho) + f(\pi) \times \rho, \tau_{min}), \tau_{max}) \tag{C.3}$$

where $f(\pi) = 0$ if i and j do not appear consecutively in π. The pheromones associated with the cuts are also similarly updated. For a pair of photos i, j in π with cut c between them:

$$\hat{\tau}_{ijc} \leftarrow min(max(\hat{\tau}_{ijc} \times (1.0 - \rho) + f(\pi) \times \rho, \tau_{min}), \tau_{max}) \tag{C.4}$$

[1]Various feasible models may be suggested but over here we aim to simply demonstrate the usefulness of learning.

C.3 Experiments and Results

Each algorithm is run 10 times on each instances with 500 iterations as the terminating criterion. To be consistent with (Atkins, 2008) we used $\theta = 4$. the other parameters were set as $\rho = 0.01$, $\tau_{min} = 0.001$ and $\tau_{max} = 0.999$. These parameters were determined from previous experiments and we do not claim here that these parameters are optimal by any means. The aim here is to simply show that the learning is effective.

The datasets that were used in (Atkins, 2008) are tested here (these are mainly small with a maximum of 12 photos) along with others generated to demonstrate the algorithms on large problems. In total, 14 datasets with photos ranging from $5 - 70$ are considered. The canvas dimensions ranged between $24 - 128$ units. The suggested relative areas and aspects ratios of the photos are given. For the new datasets, the aspect ratios were uniformly selected in the interval $[0.5, 1.5]$ so that no photo's height is much greater than its width or vice versa. This ensures that the aspect ratio is realistic. Table C.1 show the results for BRIC and BRIC-ACO. The BRIC algorithm is run with a random order where a photo is sected anew each time a new photo is appended to the layout. The algorithm BRIC-ACO-NL does not attempt to learn the photo sequence or cut information. All the results where BRIC-ACO performs the best are marked in bold face. If BRIC-ACO performs better than BRIC but not better than BRIC-ACO-NL then the results are marked in italics.

The first set of results are datasets obtained from Atkins (2008). The second set of results were synthetically generated where photo aspect ratios were restricted to the interval $[0.5, 1.5]$. The following details can be inferred from the name of a dataset, for example, $D7.32.24.0$ has 7 photos, canvas dimensions 32×24 and is instance number 0.

Table C.1: Results for BRIC and BRIC-ACO for the photo album layout problem without borders. Statistically significant results ($p = 0.05$) between the two best performing algorithms for an instance are marked in boldface.

	BRIC			BRIC-ACO			BRIC-ACO-NL		
Instance	Best	Mean	Failed	Best	Mean	Failed	Best	Mean	Failed
D5.24.32.1	5.0	5.0	0	5.0	5.0	0	5.0	5.0	0
D7.32.24.0	7.0	7.0	0	7.0	7.0	0	7.0	7.0	0
D7.32.24.1-1	7.0	7.0	0	7.0	7.0	0	7.0	7.0	0
D7.32.24.1-2	7.0	7.0	0	7.0	7.0	0	7.0	7.0	0
D12.16.48.1	12.0	12.0	0	12.0	12.0	0	12.0	12.0	0
D12.24.32.1	12.0	11.9	0	12.0	12.0	0	12.0	12.0	0
D12.32.24.1	12.0	12.0	0	12.0	12.0	0	12.0	12.0	0
D30.48.64.0	2.60E-001	2.17E-001	0	*30.0*	***27.0***	0	30.0	12.4	0
D30.64.48.1	6.21E-001	3.55E-001	0	*30.0*	***28.2***	0	8.0	5.4	0
D40.48.64.0	2.79E-002	2.05E-002	0	***37.2***	***10.4***	0	1.1	3.49E-001	0
D40.64.48.1	8.66E-002	3.74E-002	0	***9.7***	***3.7***	0	1.69	9.31E-001	0
D50.64.96.0	5.13E-003	4.06E-003	0	***49.9***	***6.1***	0	7.17E-001	1.37E-001	0
D50.96.65.1	4.37E-003	2.39E-003	0	***49.94***	***12.2***	0	2.46E-001	8.10E-002	0
D70.96.128.0	3.75E-005	2.88E-005	0	***6.3E-003***	***1.2E-003***	0	7.65E-004	2.26E-004	0
D70.128.96.1	5.06E-005	3.07E-005	0	***1.5E-003***	***6.4E-004***	0	1.29E-003	5.05E-004	0

Appendix D

SMJS: Comparison with CP-ACS

The original CP-ACO algorithm proposed by Meyer & Ernst (2004) used ACS for the ACO component of their algorithm. We used $\mathcal{MM}$AS for the ACO component in our implementations since the original Beam-ACO algorithm (Blum, 2005) used $\mathcal{MM}$AS and this variant of ACO was better suited to a hybrid with beam seach. Here, we examine CP-ACS with CP-$\mathcal{MM}$AS to determine differences in their performance.

Table D.1 shows these results. In terms of feasibility, with increasing problem size, CP-$\mathcal{MM}$AS has a clear advantage for all the problems of size 48 or more. CP-$\mathcal{MM}$AS never fails on any run. There is no clear winner when considering solution quality. However, given the feasibility results, clearly $\mathcal{MM}$AS is the preferred variant to use with CP-ACO for this problem.

Table D.1: Results of CP-ACS and CP-$\mathcal{MM}$AS for all considered instances. Statistically significant results ($p = 0.05$) are marked in boldface.

	CP-ACS				CP-$\mathcal{MM}$AS			
Instance	best	mean	sd	fail	best	mean	sd	fail
W8.1	8321	8321.00	0.00	0	8321	8321.00	0.00	0
W8.2	5818	5818.00	0.00	0	5818	5818.00	0.00	0
W8.3	4245	4245.00	0.00	0	4245	4245.00	0.00	0
W20.1	8504	**8542.80**	43.84	0	8914	9056.80	86.04	0
W20.2	5062	5076.00	14.04	0	5062	**5062.00**	0.00	0
W20.3	4312	4329.67	20.46	0	4312	**4312.00**	0.00	0
W30.1	8127	**8630.00**	286.89	0	8887	9068.50	127.67	0
W30.2	4542	**4641.20**	52.31	0	4682	4839.70	156.81	0
W30.3	4203	**4256.20**	49.31	0	4288	4364.20	48.36	0
RBG10.a	3840	3840.00	0.00	0	3840	3840.00	0.00	0
RBG16.a	2596	2596.00	0.00	0	2596	2596.00	0.00	0
RBG16.b	2094	2094.00	0.00	0	2094	2094.00	0.00	0
RBG21.9	4481	4481.00	3.38	0	4481	4481.00	0.00	0
RBG27.a.3	927	940.67	14.04	0	927	**927.10**	0.25	0
RBG27.a.15	1336	**1396.60**	28.79	0	1500	1543.60	18.54	0
RBG27.a.27	1076	1076.00	0.00	0	1076	1076.00	0.00	0
BRI17.a.3	1003	1131.13	203.22	0	1003	**1003.00**	0.00	0
BRI17.a.10	1031	1130.63	154.56	0	1031	**1031.00**	0.00	0
BRI17.a.17	1057	1057.00	0.00	0	1057	1057.00	0.00	0
RBG027a	5093	**5135.40**	23.00	0	5132	5177.70	18.51	0
RBG031a	3498	3498.00	0.00	0	3498	3498.00	0.00	0
RBG033a	3757	3757.00	0.00	0	3757	3757.00	0.00	0
RBG034a	3314	**3323.90**	39.80	0	3314	3362.00	31.10	0
RBG035a	3388	**3413.30**	35.10	0	3388	3446.10	44.41	0
RBG035a.2	3325	3325.00	0.00	0	3325	3325.00	0.00	0
RBG038a	5699	**5822.00**	95.20	0	5699	5914.90	83.45	0
RBG040a	5679	5695.30	28.50	0	5679	**5680.70**	5.74	0
RBG041a	3793	**3811.80**	35.90	0	3793	3906.30	89.41	0
RBG042a	3295	**3399.60**	59.70	0	3363	3491.20	71.08	0
RBG048a	9836	**9836.00**	-	29	9856	10019.30	89.26	0
RBG049a	13257	**13296.00**	55.20	28	13257	13401.10	72.09	0
RBG050a	12050	12066.20	37.20	0	12050	**12050.90**	5.11	0
RBG050b	-	-	-	30	12039	**12155.40**	69.65	0
RBG050c	-	-	-	30	11027	**11115.10**	65.70	0
RBG055a	6929	**6990.70**	58.60	2	6929	7045.50	64.32	0
RBG067a	10331	10460.10	68.50	21	10368	10485.00	65.51	0
RBG086a	16899	17062.10	94.00	6	16899	17028.80	85.06	0
RBG092a	12501	12516.30	26.60	27	12501	12530.10	35.77	0
RBG125a	14296	14399.00	105.10	27	14265	14383.20	98.45	0
RBG132	18524	18592.30	54.90	26	18524	18594.60	74.54	0
RBG132.2	18524	**18620.30**	110.50	24	18535	18764.50	96.63	0
RBG152	17455	17455.00	-	29	17455	17455.00	0.00	0
RBG152.2	17455	17614.30	156.60	27	17455	17505.80	64.30	0
RBG152.3	17530	17694.60	151.40	25	17458	17773.60	130.54	0
RBG172	0	-	-	30	17846	17991.70	63.56	0
RBG193	0	-	-	30	21401	21696.60	103.08	3
RBG193.2	0	-	-	30	21401	21401.30	1.41	12

Appendix E

Traveling Tournament

The travelling tournament problem consists of a double round robin tournament between a n teams (an even number). The tournament requires that a schedule be built for all the teams such that all teams play a home and away game against all other teams ($2 \times (n-1)$ rounds). The objective is to minimize the cumulative distance travelled by the teams given an $n \times n$ distance matrix between the teams. There are a number of hard constraints to take into account (1) Each team must play once every round, (2) No two teams should play consecutive games and (3) There are at most three home and away games allowed for any team.

A CP model for this problem can be specified as follows. The decision variables are $x_{i,j} \in \{-n, \ldots, -1, 1, \ldots, n\}, i \in teams, j \in rounds$. A negative value indicates an away game for a team, for example, $x_{4,7} = -3$ specifies that team 3 plays an away game in round 7 at team 4's home ground. We can specify the following constraints:

$$
\begin{array}{ll}
\forall i, j: & distinct(x_{i,j}) \\
\forall i: & x_{i,j} \neq i \wedge x_{i,j} \neq -i \\
\forall i: & linear(x_{i,j}, 0) \\
\forall i > 1, j, k: & x_{i-1,j} = k \Rightarrow x_{i,j} \neq -k \\
\forall i > 3, j: & x_{i-3,j} > 0 \wedge x_{i-2,j} > 0 \wedge x_{i-1,j} > 0 \Rightarrow x_{i,j} < 0 \\
\forall i > 3, j: & x_{i-3,j} < 0 \wedge x_{i-2,j} < 0 \wedge x_{i-1,j} < 0 \Rightarrow x_{i,j} < 0
\end{array}
$$

The model can be further strengthened by using binary variables, e.g., $y_{i,j}$, to represent home and away games. Here, the additional constraint $\forall j : linear(y_{i,j}) = n/2$ may be defined and then a mapping between $x_{i,j}$ and $y_{i,j}$ is possible: $\forall i, j : x_{i,j} > 0 \Rightarrow y_{i,j} = 0 \wedge x_{i,j} < 0 \Rightarrow y_{i,j} = 1$. In initial experiments, this model proved to be effective and feasible solutions are easily found without backtracking.

The issue that arises here is that the ACO learning model proves ineffective. In addition to the models suggested by (Uthus, Riddle and Guesgen, 2009) we have attempted three separate pheromone models which are listed below:

1. Select a team to play given the team in the previous round: O(n^2)
2. Select a team to play for the home team and current round: O(n^3)
3. Select a team to play given the home team and the team in the previous round (similar to n TSP models): O(n^3)

None of the above models are successful in learning which choices to make. The reason for this could be the option (1) is too simplistic and the learning component does not map effectively across the teams. The other options are O(n^3) and learning a large number of parameters is not effective. These reasons coupled with a large number of hard constraints may the learning component ineffective. While the CP model suggested here is effective the combination of CP-ACO is rendered ineffective and CP-Beam-ACO does not provide any improvement over CP-ACO.

References

Aarts, E. H. L., Korst, J. H. M. and van Laarhoven, P. J. M. (1997). Simulated Annealing, *in* E. H. L. Aarts and J. K. Lenstra (eds), *Local Search in Combinatorial Optimization*, Princeton University Press, pp. 91–120.

Aarts, E. and Lenstra, J. K. (eds) (2003). *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, NJ, USA.

Abramson, D., Giddy, J. and Kotler, L. (2000). High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, *In International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society, Washington, DC, USA, pp. 520–528.

Applegate, D., Bixby, R., Chvátal, V. and Cook, W. (1998). On the Solution of Traveling Salesman Problems, *Documenta Mathematica* **Extra Volume Proceedings ICM III**: 645–656.

Applegate, D. L., Bixby, R. E., Chvátal, V. and Cook, W. J. (2007). *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, Princeton, NJ, USA.

Apt, K. R. (2003). *Principles of Constraint Programming*, Cambridge University Press, Cambridge, UK.

Ascheuer, N., Fischetti, M. and Grőtschel, M. (2001). Solving the Asymmetric Travelling Salesman Problem with Time Windows by Branch-and-Cut, *Mathematical Programming* **90**: 475–506.

Atkins, C. B. (2008). Blocked Recursive Image Composition, *Proceeding of the 16th ACM international conference on Multimedia*, ACM, New York, NY, USA, pp. 821–824.

Backer, B. D., Furnon, V., Shaw, P., Kilby, P. and Prosser, P. (2000). Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics, *Journal of Heuristics* **6**: 501–523.

Ballestin, F. and Trautmann, N. (2008). An Iterated-local-search Heuristic for the Resource-constrained Weighted Earliness-tardiness Project Scheduling Problem, *International Journal of Production Research* **46**: 6231–6249.

Barnier, N. and Brisset, P. (1998). Combine Conquer: Genetic Algorithm and CP for Optimization, *In Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, Springer-Verlag, London, UK, pp. 43–46.

Bauer, A., Bullnheimer, B., Hartl, R. F. and Strauss, C. (2000). Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization, *Central European Journal of Operations Research* **8**: 125–141.

Bautista, J., Companys, R. and Corominas, A. (1988). Filtered Beam Search in Scheduling, *International Journal of Production Research* **26**: 35–62.

Bautista, J., Companys, R. and Corominas, A. (1996). Heuristics and Exact Algorithms for Solving the Monden Problem, *European Journal of Operational Research* **88**: 101–113.

Bautista, J., Pereira, J. and Adenso-Díaz, B. (2008). A Beam Search Approach for the Optimization version of the Car Sequencing Problem, *Annals of Operations Research* **159**: 233–244.

Benoist, T., Laburthe, F. and Rottembourg, B. (2001). Lagrange Relaxation and Constraint Programming Collaborative Schemes for Travelling Tournament Problems, *In Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 15–26.

Bertsimas, D. and Tsitsiklis, J. (1997). *Introduction to Linear Optimization*, Athena Scientific, New Hampshire, USA.

Blum, C. (2005). Beam-ACO: Hybridizing Ant Colony Optimization with Beam Search: An Application to Open Shop Scheduling, *Computers and Operations Research* **32**: 1565–1591.

Blum, C. and Blesa, M. (2008). Solving the KCT Problem: Large-scale Neighborhood Search and Solution Merging, *in* E. Alba, C. Blum, P. Isasi, C. León and J. A. Gómez (eds), *Optimization Techniques for Solving Complex Problems*, John Wiley & Sons, Hoboken, NJ, USA, pp. 407–421.

Blum, C. and Dorigo, M. (2004). The Hyper-Cube Framework for Ant Colony Optimization, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **3**: 1161–1172.

Blum, C., Puchinger, J., Raidl, G. and Roli, A. (2010). A Brief Survey on Hybrid Metheuristics, *In 4th International Conference on Bioinspired Optimization Methods and their Applications - BIOMA 2010*, Kluwer Academic Publishers, The Netherlands, pp. 3–16.

Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, *ACM Computing Surveys* **35**: 268–308.

Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY, USA.

Borning, A. (1981). The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, *ACM Transactions on Programming Languages and Systems* **3**: 353–387.

Brucker, P., Drexl, A., Mohring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained Project Scheduling: Notation, Classification, Models, and Methods, *European Journal of Operational Research* **112**: 3–41.

Burke, E. K., Cowling, P. I. and Keuthen, R. (2001). Effective Local and Guided Variable Neighbourhood Search Methods for the Asymmetric Travelling Salesman Problem, *In Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, Springer-Verlag, London, UK, pp. 203–212.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G. and Bonabeau, E. (2001). *Self-Organization in Biological Systems*, Princeton University Press, Princeton, NJ, USA.

Carlsson, M., Ottosson, G. and Carlson, B. (1997). An Open-ended Finite Domain Constraint Solver, *In Proceedings of the 9th International Symposium on Programming Languages: Implementations, Logics, and Programs - PLILP 1997*, Springer-Verlag, London, UK.

Carpaneto, G., Martello, S. and Toth, P. (1988). Algorithms and Codes for the Assignment Problem, *Annals of Operations Research* **13**: 193–223.

Cerny, V. (1985). A Thermodynamical approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm, *Journal of Optimization Theory and Applications* **45**: 41–51.

Christofides, N., Mingozzi, A., Toth, P. and Sandi, C. (eds) (1979). *Combinatorial Optimization*, John Wiley and Sons Ltd, Chichester, UK.

Chu, P. C. and Beasley, J. E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem, *Journal of Heuristics* **4**: 63–86.

Clements, D., Crawford, J., Joslin, D., Nemhauser, G., Puttlitz, M. and Savelsbergh, M. (1997). Heuristic Optimization: A Hybrid AI/OR Approach, *In Proceedings of the Workshop on Industrial Constraint-Directed Scheduling. In conjunction with the Third International Conference on Principles and Practice of Constraint Programming - CP97*, Vol. 1330, Springer, Heidelberg, Germanny.

Coello, C. (2002). Theoretical and Numerical Constraint-handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art, *Computer Methods in Applied Mechanics and Engineering* **191**: 1245–1287.

Congram, R. (2000). *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*, PhD thesis, School of Methematics, University of Southampton, UK.

Cook, W. (2011). The TSP page. Available from http://www.tsp.gatech.edu/.

Cook, W., Espinoza, D. G. and Goycoolea, M. (2007). Computing with Domino-Parity Inequalities for the Traveling Salesman Problem (TSP), *INFORMS Journal on Computing* **19**(3): 356–365.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001). *Introduction to Algorithms*, 2nd edn, The MIT Press, Cambridge, MA, USA.

Crauwels, H. and Oudheusden, D. V. (2003). Ant Colony Optimization and Local Improvement. Workshop of Real-Life Applications of Metaheuristics.

Danna, E., Rothberg, E. and Pape, C. L. (n.d.). Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions, *Mathematical Programmming* **102**: 71–90.

Demeulemeester, E. and Herroelen, W. (2002). *Project Scheduling: A Research Handbook*, Kluwer, Boston, MA, USA.

den Besten, M., Stűtzle, T. and Dorigo, M. (2000). Ant Colony Optimization for the Total Weighted Tardiness Problem, *Lecture Notes in Computer Science* **1917**: 611–620.

Denzinger, J. and Offermann, T. (1999). On Cooperation Between Evolutionary Algorithms and Other Search Paradigms, *IEEE Congress on Evolutionary Computation* **3**: 2317–2324.

Dincbus, M., Simonis, H. and Hentenryck, P. (1988). Solving the Car-sequencing Problem in Constraint Logic Programming, *8th European Conference on Artificial Intelligence - ECAI 88*, Pitmann Publishing, London, UK, pp. 290–295.

Dorigo, M. and Gambardella, L. M. (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions On Evolutionary Computation* **1**: 53–66.

Dorigo, M. and Stűtzle, T. (2004). *Ant Colony Optimization*, MIT Press, Cambridge, Massachusetts, USA.

Dychhoff, H. (1990). A Typology of Cutting and Packing Problems, *European Journal of Operations Research* **44**: 145–159.

Easton, K., Nemhauser, G. and Trick, M. (2003). Solving the Travelling Tournament problem: A Combined Integer Programming and Constraint Programming Approach, *Practice and Theory of AutomatedTimetabling IV* **2740**: 100–109.

Eiben, A. E. and Schippers, C. A. (1998). On Evolutionary Exploration and Exploitation, *Fundamenta Informaticae* **35**: 1–16.

Filho, G. R. and Lorena, L. A. N. (2000). Constructive Genetic Algorithm and Column Generation: An Application to Graph Coloring, *In Proceedings of APORS 2000 - The Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS*.

Fischetti, M. and Lodi, A. (2003). Local Branching, *Mathematical Programming* **98**: 23–47.

Focacci, F., Laburthe, F. and Lodi, A. (2001). Local Search and Constraint Programming. In 4th Metaheuristics International Conference - MIC2001.

Foulds, L. R. (1984). *Combinatorial Optimization for Undergraduates*, Springer-Verlag, New York, USA.

Fowler, R. J., Paterson, M. and Tanimoto, S. L. (1981). Optimal packing and covering in the plane are NP-complete, *Information Processing Letters* **12**(3): 133–137.

French, A. P., Robinson, A. C. and M.Wilson, J. (2001). Using a Hybrid Genetic Algorithm / Branch and Bound Approach to Solve Feasibility and Optimization Integer Programming Problems, *Journal of Heuristics* **7**: 551–564.

Gambardella, L. M. and Dorigo, M. (2000). An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, *INFORMS Journal on Computing* **12**: 237–255.

Gecode (2010). Gecode: Generic Constraint Development Environment. Available from http://www.gecode.org.

Gent, I. (1998). Two Results on Car Sequencing Problems, *Technical Report APES02*, University of St. Andrews, St. Andrews, UK.

Gent, I. and Walsh, T. (1999). CSPLib: A Benchmark Library for Constraints, *Technical report*, Technical report APES-09-1999. Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).

Glover, F. (1989). Tabu Search - Part I, *INFORMS Journal on Computing* **1**: 190–206.

Glover, F. (1990). Tabu Search - Part II, *INFORMS Journal on Computing* **2**: 4–32.

Gottlieb, J., Puchta, M. and Solnon, C. (2003). A Study of Greedy, Local Search and ACO for Car Sequencing Problems, *Lecture Notes in Computer Science* **2611**: 245–282.

Gravel, M., Gagné, C. and Price, W. (2004). Review and Comparison of Three Methods for the Solution of the Car-sequencing Problem, *Journal of the Operational Research Society* **56**: 1287–1295.

Hentenryck, P., Simonis, H. and Dincbus, M. (1992). Constraint Satisfaction using Constraint Logic Programming, *Artificial Intelligence* **58**: 113–159.

Hentenryck, P. V. (1989). *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, Massachusetts, USA.

Hillier, F. S. and Lieberman, G. J. (2005). *Introduction to Operations Research*, 8 edn, McGraw Hill, New York, NY, USA.

Hopper, E. and Turton, B. C. H. (2001). An Empirical Investigation of Metaheuristics and Heuristic Algorithms for 2D Packing Problem, *European Journal of Operational Research* **128**: 34–57.

ILOG (2007). ILOG Constraint Programming. Available from http://www.ilog.com/products/cp/.

Johnson, T. (1967). *An Algorithm for the Resource-constrained Project Scheduling Problem*, PhD thesis, Sloan School of Management, MIT, USA.

Joslin, D. E. and Clements, D. P. (1998). Squeaky Wheel Optimization, *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, American Association for Artificial Intelligence, Madison, WI, USA, pp. 340–346.

Khichane, M., Albert, P. and Solnon, C. (2008). CP with ACO, *Lecture Notes in Computer Science* **5015**: 328–332.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing, *Science* **220**: 671–680.

Kis, T. (2004). On the Complexity of the Car Sequencing Problem, *Operations Research Letters* **32**: 331–335.

Klau, G., Ljubi, I., c Moser, Mutzel, P., Neuner, P., Pferschy, U. and Weiskircher, R. (2004). Combining a Memetic Algorithm with Integer Programming to Solve the Prize-collecting Steiner Tree Problem, *Lecture Notes in Computer Science* **3102**.

Kostikas, K. and Fragakis, C. (2004). Genetic Programming Applied to Mixed Integer Programming, *Lecture Notes in Computer Science* **3003/2004**: 113–124.

Krasnogor, N. and Smith, J. (2005). A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues, *IEEE Transactions on Evolutionary Computation* **9**: 474–488.

Kuhn, H. W. (2010). The Hungarian Method for the Assignment Problem, *in* M. Jünger, T. M. Liebling., D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi and L. A. Wolsey (eds), *50 Years of Integer Programming 1958-2008*, Springer Berlin Heidelberg, pp. 29–47.

Larrañaga, P. and Lozano, J. A. (2002). *Estimation of Distribution Algorithms*, Kluwer Academic Publishers, Cambridge, MA, USA.

Leguizamon, G. and Michalewicz, Z. (1999). A New Version of Ant System for Subset Problems, *IEEE Congress of Evolutionary Computation* **55**: 1459–1464.

Lewis, H. R. and Papadimitrious, C. H. (1981). *Elements of the Theory of Computation*, Prentice-Hall, New Jersey, USA.

López-Ibáñez, M. and Blum, C. (2008). Beam-ACO Based on Stochastic Sampling: A Case Study on the TSP with Time Windows, *Technical Report LSI-08-28*, Department LSI, Univeristat Politècnica de Catalunya.

López-Ibáñez, M., Blum, C., Thiruvady, D., Ernst, A. T. and Meyer, B. (2009). Beam-aco Based on Stochastic Sampling for Makespan Optimization Concerning the TSP with Time Windows, *Lecture Notes in Computer Science* **5482**: 97–108.

Marriott, K. and Stuckey, P. (1998). *Programming With Constraints*, MIT Press, Cambridge, Massachusetts, USA.

Martello, S., Monaci, M. and Vigo, D. (2003). An exact approach to the strip-packing problem, *INFORMS Journal on Computing* **15**(3): 310–319.

Meyer, B. and Ernst, A. (2004). Integrating ACO and Constraint Propagation, *Lecture Notes in Computer Science: Ant Colony, Optimization and Swarm Intelligence* **3172/2004**: 166–177.

Michalewicz, Z. and Fogel, D. (2004). *How to Solve It: Modern Heuristics*, 2 edn, Springer-Verlag, Berlin, Heidelberg, Germany.

Milano, M., Ottosson, G., Refalo, P. and Thorsteinsson, E. S. (2001). Global Constraints: When Constraint Programming Meets Operation Research, *INFORMS Journal on Computing, Special Issue on the Merging of Mathematical Programming and Constraint Programming* .

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA.

Mladenovic, N. and Hansen, P. (1997). Variable Neighborhood Search, *Computers and Operations Research* **24**: 1097–1100.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*, Dover, New York, USA.

Parrello, B., Kebat, W. and Wos, L. (1986). Job-shop Scheduling using Automated Reasoning: a Case Study of the Car Sequencing Problem, *Journal of Automated Reasoning* **2**: 1–42.

Pelikan, M. (2005). *Heirarchical Bayesian Optimization Algorithm*, Springer-Verlag, Berlin, Germany.

Pelikan, M., Goldberg, D. E. and Cantú-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm, *in* W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela and R. E. Smith (eds), *In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-99*, Vol. I, Morgan Kaufmann Publishers, San Fransisco, CA, USA, pp. 525–532.

Perron, L. and Shaw, P. (2004). Combining Forces to Solve the Car Sequencing Problem, *CPAIOR-04*, Vol. 3011, Springer-Verlag, pp. 225–239.

Pinedo, M. L. (2005). *Planning and Scheduling in Manufacturing and Services*, Springer, New York, USA.

Plateau, A., Tachat, D. and Tolla, P. (2002). A Hybrid Search Combining Interior Point Methods and Metaheuristics for 0-1 Programming, *International Transactions in Operational Research* **9**: 731–746.

Puchinger, J. and Raidl, G. R. (2004). An Evolutionary Algorithm for Column Generation in Integer Programming: an Effective Approach for 2D Bin Packing, *Lecture Notes in Computer Science* **3242**: 642–651.

Puchinger, J. and Raidl, G. R. (2005). Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification, *Lecture Notes in Computer Science* **3562**: 41–53.

Puchinger, J., Raidl, G. R. and Koller, G. (2004). Solving a Real-World Glass Cutting Problem, *Lecture Notes in Computer Science* **3004**: 162–173.

Raidl, G. and Feltl, H. (2004). An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem, *in Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, ACM, New York, NY, USA, pp. 990–995.

Raidl, G. R. (1998). An Improved Genetic Algorithm for the Multiconstrained 0-1 Knapsack Problem, *in* D. B. Fogel (ed.), *In Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, IEEE Press, pp. 207–211.

Regin, J. and Puget, F. (1997). A Filtering Algorithm for Global Sequencing Constraints, *Lecture Notes in Computer Science* **1330**: 32–46.

Reinelt, G. (2007). The TSP library: TSPLIB. Available from http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*, 2nd edn, McGraw-Hill Higher Education.

Roli, A., Blum, C. and Dorigo, M. (2001). ACO for Maximal Constraint Satisfaction Problems. In 4th Metaheuristics International Conference - MIC2001.

Roos, C., Terlaky, T. and Vial, J.-P. (2006). *Interior Point Methods for Linear Optimization*, Springer, New York, USA.

Rothberg, E. (2007). An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions, *INFORMS Journal on Computing* **19**(4): 534–541.

Ruml, W. (2001). Incomplete Tree Search using Adaptive Probing, *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence - IJCAI-01*, AAAI, Menlo Park, CA, USA, pp. 235–241.

Savelsbergh, M. W. P. (1985). Local Search in Routing Problems with Time Windows, *Annals of Operations Research* **4**: 285–305.

Schwindt, C., Neumann, K. and Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources*, Springer, Berlin, Germany.

Shaw, P., Backer, B. D. and Furnon, V. (2002). Improved Local Search for CP Toolkits, *Annals of Operations Research* **115**: 31–50.

Singh, G. and Ernst, A. T. (2011). Resource Constraint Scheduling with a Fractional Shared Resource, *Operations Research Letters* **In Press, Corrected Proof**: –.

Sipser, M. (2005). *Introduction to the Theory of Computation*, 2 edn, Course Technology.

Smith, B. (1997). Succeed-first or Fail-first: A Case Study in Variable and Value Ordering Heuristics, *I: Proceedings of The Third International Conference on the Practical Applications of Constraint Technology - PACT-97*, Practical Applications Company, Blackpool, UK, pp. 321–330.

Solnon, C. (2002). Ants can Solve Constraint Satisfaction Problems, *IEEE Transactions on Evolutionary Computation* **6**: 347–357.

Solnon, C. (2008). Combining two Pheromone Structures for Solving the Car Sequencing Problem with Ant Colony Optimization, *European Journal of Operational Research* **191**: 1043–1055.

Solnon, C., Cung, V. D., Nguyen, A. and Artigues, C. (2008). The Car Sequencing Problem: Overview of State-of-the-art Methods and Industrial Case-study of the ROADEF'2005 Challenge Problem, *European Journal of Operational Research* **191**: 912–927.

Steele, G. (1980). *The Definition and Implementation of a Computer Programming Language Based on Constraints*, PhD thesis, Department of Electrical Engineering and Computer Science, Cambridge, MA, USA.

Stűtzle, T. and Hoos, H. H. (2000). MAX-MIN Ant System, *Future Generation Computer Systems* **16**: 889–914.

Stinson, J. P., Davis, E. W. and Khumawala, B. M. (1978). Multiple ResourceConstrained Scheduling using Branch and Bound, *AIIE Transactions* **10**(3): 252–259.

Sutherland, I. E. (1964). Sketchpad a Man-machine Graphical Communication System, *In Proceedings of the SHARE design automation workshop - DAC '64*, DAC '64, ACM, New York, NY, USA, pp. 6.329–6.346.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, MIT Press, Cambridge, MA, USA.

Talukdar, S., Baeretzen, L., Gove, A. and de Souza, P. (1998). Asynchronous Teams: Cooperation Schemes for Autonomous Agents, *In Proceedings of the International Joint Conference on Artificial Intelligence IJCAI*, Vol. 4, pp. 295–321.

Thiruvady, D., Blum, C., Meyer, B. and Ernst, A. T. (2009). Hybridizing Beam-ACO with Constraint Programming for Single Machine Job Scheduling, *Lecture Notes in Computer Science* **5818**: 30–44.

Thiruvady, D., Meyer, B. and Ernst, A. (2008). Strip packing with hybrid aco: Placement order is learnable, *IEEE Congress on Evolutionary Computation*, IEEE press, pp. 1207–1213.

Thompson, P. and Psaraftis, H. (1993). Cycle Transfer Algorithm for Multivehicle Routing and Scheduling Problems, *Operations Research* **41**: 935–946.

Uthus, D. C., Riddle, P. J. and Guesgen, H. W. (2009). An Ant Colony Optimization Approach to the Traveling Tournament Problem, *In Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09*, ACM, New York, NY, USA, pp. 81–88.

Valente, J. M. S. and Alves, R. A. F. S. (2008). Beam Search Algorithms for the Single Machine Total Weighted Tardiness Scheduling Problem with Sequence-Dependent Setups, *Computers and Operations Research* **35**(7): 2388–2405.

Valls, V., Quintanilla, S. and Ballestin, F. (2003). Resource-constrained Project Scheduling: A Critical Activity Reordering Heuristic, *European Journal of Operational Research* **149**: 282–301.

van Hoeve, W., Pesant, G., Rousseau, L. and Sabharwal, A. (2006). Revisiting the Sequence Constraint, *Lecture Notes in Computer Science* **4204**: 620–634.

Vasquez, M. and Hao, J.-K. (2001). A Hybrid Approach for the 01 Multidimensional Knapsack Problem, *In Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI*, AAAI, Menlo Park, CA, USA, pp. 328–333.

Vasquez, M. and Vimont, Y. (2005). Improved Results on the 0-1 Multidimensional Knapsack Problem, **165**: 70–81.

Woodruff, D. L. (1999). A Chunking Based Selection Strategy for Integrating Metaheuristics with Branch and Bound, *in* S. V. et al. (ed.), *In Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, The Netherlands, pp. 499–511.