In this chapter I survey reported computational experience with DantzigWolfe decomposition. My intention is threefold—to identify the computational techniques and strategies that have been used to implement the algorithm, so that my implementation can draw on the best practice of previous work; to describe the practical results obtained so far with decomposition on real applications; and to identify opportunities for us to make further progress. My contribution here is a critical review of previous practical work, drawing out issues for a new implementation and opportunities for further advances. My survey is in the main chronological, but I have identiﬁed ﬁve distinct phases of work into which the chapter is structured. The ﬁrst and largest section covers the period up until the late 1970s of the ﬁrst practical experiences with decomposition. This period is characterised by a wealth of ideas, software implementations and problems tackled, but little evolutionary improvement in the computational development. The second section considers the major contribution made by Ho, Loute et al over a period of ten years. By drawing on their own experiences and other work, they developed a series of second generation software that removed many of the doubts about decomposition, but ﬁnally showed that it cannot compete with mainstream LP methods on most problems. Their work had just two weaknesses—a lack of a ﬁnal systematic evaluation of the algorithm, and a tight integration of their software to certain LP software that prevented re-useandimprovementoftheircodeoncetheLPcodehadbeensuperseded. The third section reports on the limited experiences with parallel implementations of the decomposition method: there is much scope here for further work. During the 1990s, the innovation of interior point methods reached decomposition with some positive eﬀect, with a signiﬁcant amount of work done at Logilab in Geneva, and this is reported in section fmy.

SURVEY OF COMPUTATIONAL EXPERIENCE

Finally I mention the most recent topic of interest: that of incorporating decomposition into a branch and bound framework in order to solve integer programming problems. I end the chapter by drawing together and discussing issues for a computational implementation, and noting opportunities for further work.

First Experiences

Beale, Hughes and Small reported the ﬁrst practical application of DantzigWolfe decomposition in 1965, applying decomposition to an oilﬁeld operations problem [BHS65]. In many ways their study remains the most useful of all those implementing Dantzig-Wolfe decomposition, for it contained innovative ideas for implementing the algorithm together with positive practical results; the authors were always able to come up with new ideas to circumvent poor performance. The algorithm was implemented as an internal extension to the commercial lp/90/94 mathematical programming software. The software had a theoretical capacity of 100 blocks, 100,000 constraints in total, 1000 constraints in any block, and an unlimited number of variables. The major innovation was to retain the coeﬃcients of the global constraints (Ak) in the pricing problems—as non-binding constraints with zero RHS. This had three apparent advantages: 1. The modiﬁed objective function of each pricing problem, c0 k = ck − πAk, could be calculated directly in the underlying LP optimiser as a linear combination of these constraints. (A simple modiﬁcation to the internal simplex routine FORMC allowed an easy implementation.) 2. The proposal coeﬃcients for extreme point proposals could be read directly from the slack values of the global constraints; or, in the case of extreme ray proposals, directly from the matrix coeﬃcients of these constraints, as represented in the current basis. 3. The reconstruction of the full primal solution could be implemented easily—setting the RHS of the global constraints in each of the pricing problems to be the computed optimal allocation and solving the pricing problems with the constraints enforced. Exploiting the underlying simplex optimiser to perform this additional work required by the decomposition algorithm greatly simpliﬁed the additional programming. However, the authors’ claim that the resulting implementation was ‘more eﬃcient’ cannot be left unchallenged; whilst the amount of work required in the decomposition-speciﬁc routines was certainly reduced, itwasnotshownthatthisoutweighedtheextraworkrequiredbythesimplex routines to carry the non-binding constraints.

FIRST EXPERIENCES

was noted that optimisation of the master problem and pricing problems could be resumed from the advanced bases obtained at the last cycle. The paper also addressed the three open algorithmic issues of proposal initialisation, proposal generation and proposal management. At the start, a representative range of proposals was computed from each block by exploiting certain problem speciﬁc knowledge. (Each proposal had a natural interpretation as an activity vector for one of the oilﬁelds.) Multiple proposal generation was used—that is, many proposals generated from one pricing problem per cycle. Proposals were generated at a certain frequency of pricing problem pivots, using a dynamic frequency—high at the start of the algorithm, to try to quickly build up the representation of each block in the master problem, then low, to prevent the master growing too large, and then high again towards the end of the algorithm. The actual frequency was controlled manually during computation from the on-line card reader based on many experimental runs. Whether a similar strategy could be applied without such detailed experiments appears unlikely. At the beginning of the algorithm, optimal proposals were intentionally suppressed, as it was found they actually slowed overall progress by misleading the master problem. Dantzigdescribedthreestrategiesfordecidinghowlongproposalsshould persist in the master problem—retain all proposals; delete all non-basic proposalsimmediately(apartfromthosejustgenerated); orkeepproposalsuntil a certain proposal capacity is reached, and then purge those proposals with the worse reduced costs [Dan63]. Beale, Hughes and Small chose the ﬁrst option: they felt that the beneﬁt of retaining coverage of the feasible region in the master problem outweighed the extra work implied by accumulating proposals. The best reported result, on a problem of 450 constraints, found the decomposition method performed marginally better than the rival simplex method. The authors claimed that this comparison was slightly unfair to the simplex method, as no tuning had been carried out for simplex, but this overlookedthefactthatthesimplexmethodingeneralhadalreadybeensubject to much reﬁnement, whereas this was one of the ﬁrst real applications of decomposition. Nevertheless, some of the more problem speciﬁc reﬁnements to the decomposition method did prove crucial; notably the generation of initial proposals and the dynamic proposal generation frequency. The work of Beale, Hughes and Small was important because it established a benchmark implementation for Dantzig-Wolfe decomposition. The computational techniques they used were not matched or improved on until rediscovered by Ho and Loute over a decade later. Orchard-Hays described an implementation of decomposition as an internal addition to the optimal mathematical programming software for the CDC 6000 [OH68]. He suggested an alternative criterion for generating proposals: generating one from the ﬁrst solution to pass the reduced cost test, followed by the next ﬁve for which the objective value (i.e., the reduced

SURVEY OF COMPUTATIONAL EXPERIENCE

cost of the corresponding proposals adjusted by a constant) improved by a stated percentage. At unbounded solutions, generate both an extreme point proposal and an extreme ray proposal. One application, to a logistics problem, was reported. Convergence was slow, with a long tail of near optimal solutions but for which the ﬁnal optimality gap could not be closed—asymptotic convergence. Proposals were completely dense, which had the eﬀect of reducing the simplex iteration speed when solving the master [OH72]. Decomposition was not competitive with alternative methods. Summarising his practical experience up to 1973, Orchard-Hays said:

[ ... ] in two decades, while LP capabilities have ﬂmyished in scope, power, generalisation and application, decomposition remains more a principle than a practical method. [ ... ] I know of not a single instance of the use of general decomposition in a routine, production environment such has been commonplace with general LP for over a decade. [OH73]

Kutcher reported an implementation of decomposition using ICL’s xdla LP software, and the application of it to an economic planning model for agriculture in Mexico [Kut73]. Like Beale et al, he found that the prices ﬂuctuate wildly in the early stages of the algorithm, and, like OrchardHays, he observed asymptotic convergence. He experimented by varying the degree of decomposition, and found that a ﬁner grain decomposition led to fewer cycles (I return to this issue in section 5.9, p. 112). Abreakthroughinsoftwarecamein1974, whentheﬁrstimplementations of decomposition using IBM’s mpsx mathematical programming software appeared. mpsx allowed algorithms to be written that called into mpsx’s own routines, by using mpsx’s pl/i based Extended Control Language. Williams and Redwood reported an mpsx implementation decomp for a food blending application of 1800 constraints and 3200 variables in total [WR74].1 The same problem was solved repeatedly in a production environment with diﬀerent data instances. Two block structures were exploited, one based on food products and one on time periods. The product decompositiongaverisetoproposalswithanaturalinterpretationasrecipes, which allowed recipes generated on earlier runs to be used as initial proposals on later runs. Other features of the implementation worth noting are that proposals were generated once the reduced cost fell below an apparently arbitrary value of −0.2, and that the algorithm was halted once the improvement in the overall objective function from one cycle to the next fell below 0.5%. Observe that either feature may have concealed numerical instability at the expense of potentially early, non-optimal termination of the

1In the same year Ho and Manne reported an implementation of nested Dantzig-Wolfe decomposition using mpsx and pl/i [HM74].

FIRST EXPERIENCES

algorithm, perhaps far from optimal, and although the dual bound would have provided a measure of the quality of the solution it was not reported. In comparison to solving the problem with the mpsx simplex method directly, the time based decomposition took half the time to get a solution within 0.28% of optimal, while the product based decomposition had little if any advantage over the simplex method. The authors noted that while the decomposition method was accelerated by the strategies they used, the crucial feature of the simplex method was the ability to save and restore bases from one run to the next, a feature that could not really be matched by decomposition despite re-using recipes. A number of practical experiences were summarised in the oft cited book of Dirickx and Jennergren [DJ79]. They reported that in general the greater number of proposals are used to initialise the algorithm, the faster the solution can be found. It is clear that if work can be done outside the algorithm, then the algorithm will run more quickly. But if a representative set of proposals can be generated cheaply or proposals generated on other data instances can be re-used, then the beneﬁt is real. On the management of proposals in the master, they quoted Schiefer, who suggested deleting proposals once they had been non-basic for a certain number of iterations, such as 15. Their general assessment was that the decomposition applications reported had ‘not been very positive.’ The method was ‘time consuming and cumbersome [and] converges only slowly towards an optimal solution.’ Echoing Orchard-Hays’ 1973 comment, they saw that no general DantzigWolfe software was available; each study started by building its own from scratch, and thus there was little scope for evolutionary improvement. Their conclusion was that though in some cases decomposition may be the only algorithmic choice where the problem is too large to be solved in one piece, if a standard LP code was available that can handle the given problem, it should be used in preference to a decomposition code.

Certainly, the opinion of the state of the art in decomposition towards the end of the 1970s was one of disillusionment. A lot of research had been carried out, often focused on solving a particular application. There was little re-use or evolutionary improvement of software, and no systematic study of the algorithm with its various strategies on a variety of problems. Dantzig summarised the deﬁciencies of the research:

The problem has been that [ ... ] the only ﬁnal criterion can be systematic experimentation with representative models. Clearly, for such experimentation to give meaningful and reliable results, the implementation must be sophisticated and the test problems large enough to give a guide to real problem behavimy. [Dan77]

URVEY OF COMPUTATIONAL EXPERIENCE

The Advanced Implementations of Ho and Loute

During the 1970s and 1980s, Ho, Loute, and others published a series of papers which described a second generation implementation and evaluation of Dantzig-Wolfe decomposition. Ho’s interest sprang from one in nested Dantzig-Wolfe decomposition, an idea originally proposed (though often overlooked) in one of Dantzig’s and Wolfe’s original papers, and ﬁrst investigated in depth by Glassey and Ho and Manne [DW60, Gla73, Ho74, HM74]. Nested Danzig-Wolfe decomposition is a recursive application of DantzigWolfe decomposition, and some of the ideas and experience I quote below, though applicable to ordinary (two-level) decomposition, were originally made with reference to nested decomposition. Loute meanwhile had originally investigated the exploitation of block structure in the simplex algorithm itself, without the need for reformulation, by the use of block factorisation techniques [HL80]. Already by 1978, Ho, in the context of nested decomposition, had deﬁned a set of fmy parameters to control multiple proposal generation, which formalised and uniﬁed the various strategies described in the previous section [Ho78]. The ﬁrst proposal is generated at the ﬁrst feasible solution that meets the reduced cost criterion. Proposals are then generated every FREQ simplexiterations, orwheneverthereducedcostisimprovedby PERC%, until TERM proposals have been generated or the problem is optimal, and only the ﬁnal MAX proposals are submitted to the master problem. Suggested ranges and default values are MAX: 1–10, default 5; FREQ: 5–15, default 5; PERC: 0.1–1.0%, default 0.1% [Ame81]. Ho said that one should aim for rapid sub-optimal proposal generation in the early cycles, and complete optimal proposal generation in the later cycles. If the pricing problem becomes unbounded so that an extreme ray proposal is generated, further proposals can be generated (up to MAX) by ﬁxing the unbounded pricing problem variable and re-optimising. Experience showed that there were often many unbounded extreme rays in close vicinity, and it may otherwise have taken one cycle to eliminate each ray. Ho also identiﬁed data structures, the LP software, and the proposal management as three major areas on which the eﬃciency of the implementation depends. Ho’s and Loute’s ﬁrst study of two-level decomposition was a preliminaryinvestigationintosolvinglargemulti-nationalmulti-periodenergymodels [HLSvdV79]. They used a decomposition code decomp, originally written by Carlos Winkler at Stanford’s System Optimisation Laboratory (SOL) and based on John Tomlin’s revised simplex code lpm1 [HS89, Tom75b]. Thisenabledthemtoexploitlow-leveleﬃciencies, suchasdirectlymodifying the internal simplex routine FORMC to automatically calculate the modiﬁed pricing problem objective function (following [BHS65]).

THE ADVANCED IMPLEMENTATIONS OF HO AND LOUTE

Themodelunderinvestigationhadupto52,000rowsand65,000columns in its largest form, although only small instances of it were solved for the report. They argued that decomposition would win on large structured problems, basedontheobservedworkloadofdecomposition, whichincreased linearly with problem size, as compared to the simplex method, where the workload increased cubically. Ho’s and Loute’s main contribution was the decompsx implementation and associated experience [HL81, HL83]. decompsx was implemented using the Extended Control Language and LP routines of mpsx/370, at that time state of the art and the only sophisticated LP software to provide a programming interface [BGHR77]. In fact, various mpsx routines were modiﬁed, to implement a similar low-level interface as had been used with the earlier decomp code. decompsx incorporated many techniques developed piecemeal in the past—for example, the data handling scheme of [BHS65], and Ho’s uniﬁed controls for proposal generation. They also introduced some new ideas, such as partial cycles: they noted that provided at least one proposal was generated each cycle, there was no need to solve all pricing problems on each cycle, thus saving on problem setup time. In practice they skipped a problem if it had failed to generate a proﬁtable proposal on the last cycle. They developed a simple input format: a single MPS ﬁle in which the master problem and pricing problems were listed consecutively. The matrix scaling strategy of mpsx had to be augmented for decomposition, as experience showedthatproposalstendedtocontainelementswhichdiﬀeredsigniﬁcantly inmagnitudefromtheunityelementintheconvexityrow, forwhatthe mpsx scaling was insuﬃcient to handle. Instead, each pricing problem was given a problem scaling factor, used to scale all proposals generated from that pricing problem [ALR81]. The initial experience comprised a test set of 12 problems from a variety of applications such as forestry management and multi-national dynamic energy planning, with up to 6000 rows, 12,000 columns, 9 pricing problems, and with density 0.04–1.2%. All tests were conducted with the default strategies, and the problems were solved to within 0.01% of optimality. The problems took on average 27 cycles to solve, the largest up to 128 cycles, but all but two under 30 cycles. The largest problem took 72 minutes to solve, but the average time was 18 minutes. Of the total CPU time, 15–36% was taken up by the high level decomposition routines, and the rest by the low level mpsx LP routines. The authors thus proclaimed decompsx a robust and eﬃcient decomposition solver, meaning that it could successfully solve a variety of problems, and that the time spent in the higher level routines as regards the mpsx LP routines was relatively small. However, compared to the simplex method implemented in mpsx, decompsx always came out worse, in many cases taking twice as long as mpsx. All problems were solved to a relatively tight tolerance in a relatively

SURVEY OF COMPUTATIONAL EXPERIENCE

small number of cycles, in contrast to the accepted wisdom that decomposition yields slow or asymptotic convergence. Ho noted that for problems that don’t converge quickly, one of two behavimys could be observed—they either become asymptotic, or exhibit symptoms of ill-conditioning [Ho84]. Asymptoticconvergence maybeattributedtonumericalinaccuracypreventing the duality gap from closing. The error can be standardised by inverting before generating each proposal, but this can be expensive, especially if multiple proposal generation is used. Ho suggested (but did not implement) a test to calculate the error, so that an invert can be carried out when the error grows large. However, in decomposition there is really no comparable feature to the invert in the revised simplex method which periodically resets the computation with the original data [Ho87]. In an assessment of their work, Ho and Loute made three major points. Firstly, echoing Dirickx and Jennergren, they claimed that decomposition could not compete with mainstream LP techniques, but could only extend the capability of mainstream LP techniques by being applied solve larger problems than would otherwise be possible to solve. Secondly, echoing Dantzig, they claimed that for a deﬁnitive evaluation of decomposition, it is necessary to implement decomposition at the level of state of the art LP software. Thirdly, to promote understanding of decomposition, they said that decomposition must be made available as simple to use software, ideally as part of mainstream commercial LP codes. Ho and Loute’s work was the ﬁrst to really build on previous work, and at the same time to highlight its weaknesses: they incorporated various data handling techniques and computational strategies devised by previous studies and developed a simple set of parameters for controlling them. They realised the importance of using sophisticated LP software, and exploited the LP subroutine libraries that had become available. Unfortunately they did not consider the relative merits of a tight integration with the underlying LP software as compared to a clear interface; by opting for the former, with smyce code modiﬁcations to the LP software, they prevented easy porting of their decomposition routines to other LP software, with the result that their code is no longer in use. Ho and Loute evaluated their implementations on a suﬃcient variety of problems to validate decomposition as a capable and robust problem solver. Their results shifted the focus of decomposition from competing with mainstream LP methods to standing as a method in its own right, oﬀering a solution method for a small but signiﬁcant number of problems. However, despite their original intention, they did not perform a systematic study of the algorithm, and its diﬀerent computational strategies, on a comprehensive set of problems, and in this respect fell short of a deﬁnitive study of decomposition.

PARALLEL IMPLEMENTATIONS

Parallel Implementations

The decomposition method naturally lends itself to a coarse grained distributed memory parallelism based on assigning the pricing problems to differentprocessorsandsolvingthemsimultaneously.2 Atthesimplestlevelthe master problem and pricing problems reside on separate processors; when there are more problems than processors, each processor may hold several problems which may be solved locally in sequence. Mirroring the algorithm, control is maintained by the process handling the master problem: it sends outpricestostartuptheslaveprocesses; theslaveprocessessolvethepricing problems and send the generated proposals back to the master processes. Two measures are widely used to assess the performance of a parallel application in comparison to a serial application performing the same task. Let Tp be the time taken to perform the application when p processors are available. The speed-up, Sp, is the performance improvement ratio

Sp = T1/Tp

The eﬃciency, Ep, is the ratio of the speed-up to the number of processors

Ep = Sp/p

In general, I would hope for near-linear speed-up, or, equivalently, an eﬃciency close to one. The ﬁrst parallel implementation of the decomposition method was the prototype decompar by Ho et al [HLS88, HS89], based on the serial decomposition software decomp used by Ho and Loute. decomp, and its underlying LP software lpm1, were by now over ﬁfteen years old, and much inferior to other LP software available. The authors no doubt chose it due to the availability of complete smyce code and their familiarity with it, but it could not be used for practical computational work. Indeed, the maximum problem size was very restricted, with at most 10 blocks, each with 400 rows, 1000 columns and 10,000 non-zeros; and up to 99 global rows. When implemented in parallel, a new area of ﬂexibility is opened up in the algorithm: the order in which the problems are solved on each cycle, and whether all problems are solved on every cycle [HLS88]. Five strategies may be distinguished by the action taken in a typical cycle:

Basic strategy Solve all pricing problems, then solve the master problem.

First pricing problem strategy Start solving all pricing problems, then as soon as one ﬁnishes, solve the master problem.

2Such an implementation is described as MIMD—multiple instructions (meaning the processes follow diﬀerent instructions or code) multiple data (meaning the processes operate on diﬀerent data).

SURVEY OF COMPUTATIONAL EXPERIENCE

First proposal strategy Start solving all pricing problems, then as soon as one proposal is generated, solve the master problem. Instant feedback strategy Allnewinformationisactedonimmediately— all pricing problems and master constantly being solved, send out proposal from each pricing problem after every pivot, and send out prices from master after each pivot. Accelerated feedback strategy Wheneverapricingproblemﬁnishes, send out proposals generated, check for new prices, and start solving again as soon as new prices found. Whenever the master problem ﬁnishes, send out new prices, check for new proposals, and start solving again as soon as any found. The latterstrategies aimed for less idle time thanthe basic strategy, with the instant feedback strategy taking this aim to its logical conclusion; however, they result in computation being based on sub-optimal information, and so smaller steps taken by each process, with slower overall convergence. The accelerated feedback strategy was developed to strike a compromise between thesetwofactors, asaresultofexperiencewiththeinstantfeedbackstrategy. Some results were reported on tiny test problems—up to 700 rows and 1250 columns in total—and so can only be taken as validating the implementation rather than providing any real evidence as to the eﬃciency of it and with the diﬀerent computational strategies. The number of processors used varied (one per pricing problem) from 5–11, giving eﬃciencies in the range 0.5–1.0, and 0.71 on average, although a potentially major complicating factor here was the lack of disk I/O required by decompar (each problem held in the memory of a dedicated processor) as compared to decomp (each problem written to disk between cycles), which was not investigated. The results indicated that the basic strategy was faster than the ﬁrst problem strategy, which in turn was faster than the ﬁrst proposal strategy, but the accelerated feedback strategy was on average 1.3 times faster than the basic strategy. However, doubts must remain about the applicability of the results. Ho and Gnanendran reported a more sophisticated parallel implementation decube in 1993 [GH93]. Equally importantly, more extensive testing is reported with larger problems, although many of the problems were ﬁrst solved ten years previously. decube was an MIMD implementation for the iPSC/2 hypercube, and, as with decompar before, each problem was assigned to a dedicated processor. A set of parameters (K,k,p) were introduced for describing the various strategies for sequencing the problems solved on a given cycle—the master problem was solved as soon as K proposals have been submitted from all pricing problems in total, or k proposals received from each pricing problem, or p pricing problems terminated.

THE USE OF INTERIOR POINT METHODS

decube was applied to the complete set of problems used earlier by Ho and Loute, together with much larger instances (up to 30,000 rows) of the MRP problems from the same set [HL83]. Making a somewhat questionable comparison with mpsx/370 (much better would have been to compare decube with its own internal LP software, lpm1, to minimise discrepancies attributable to software diﬀerences), decube achieved parity at about 12,000 rows, and beat mpsx by factors of fmy and six on the two largest problems. No account was given of other factors that may have inﬂuenced the comparison. An improvement of 20% was attributed to the load balancing ideas, but little raw data was provided to assess this claim. Entriken made an excellent study of nested Dantzig-Wolfe decomposition, including a sophisticated parallel implementation and comprehensive practical evaluation [Ent89]. Of particular relevance to us is his description of the implementation, which I draw on in chapter 5, and his methodology for testing. His tests, on a particularly wide range of problems culled from other studies, had fmy explicit aims: to evaluate parallel nested decomposition in comparison to standard LP methods; to investigate algorithm performance under diﬀerent parameter settings; to extrapolate the performance results beyond the test set; and to identify limitations of the code and areas for improvement. It was the most comprehensive evaluation of a decomposition method made in forty years. The results obtained were also positive, showing both that nested decomposition itself provided a speed-up over the simplex method for larger problems—up to ten times in cases—and naturally parallel nested decomposition provided further speed-up. By contrast, the practical work for plain two-level decomposition in parallel has been nowhere near as comprehensive. There have been some very good ideas, and recently some further work using interior point methods (see below), but no state of the art implementation.

The Use of Interior Point Methods

The idea of exploiting structure with interior point methods has seen much interest, and in 1991 Kim and Nazareth published an excellent paper which proposed the idea of using interior point methods (IPM) within the context of the decomposition method [KN91]. The idea might at ﬁrst seem slightly odd, since decomposition is a specialised variant of the simplex algorithm, and makes use of the extreme point and ray solutions generated by the simplex method when solving the pricing problems. But as I mentioned in section 3.4.3, proposals can be generated from non-extreme points or rays of the pricing problems. Kim and Nazareth used IPM to solve the pricing problems, retaining the simplex method for the master problem. They argued that when the pricing problems are degenerate, there may be many extreme points in the vicinity

SURVEY OF COMPUTATIONAL EXPERIENCE

of the extreme point required by the master problem to represent the optimal, or indeed an intermediate, solution. Thus many cycles are expanded generating similar proposals to take tiny steps towards the optimal solution of the master. Further, numerical problems may redouble the problem by misrepresenting extreme points of the pricing problem. Conversely, interior points would be less susceptible to the exact combinatorial structure of the pricing problem (see [KN91] for a numerical example). A limited amount of practical work was also conducted (using random problems, which is often not the best test for practical optimisation codes; see [CDM78]) which showed that the IPM variant always used fewer cycles than the standard simplex variant. However, much greater work and time was required each cycle by the IPM variant to solve the pricing problems, as each pricing problem had to be solved from scratch. In the simplex variant the optimal basis for each pricing problem from the previous cycle can be exploited to give an advanced start. Other authors considered using IPM to solve just the master problem as part of the decomposition method. Here, the argument in favmy is that the dual solution (the prices) obtained by the IPM will give better guidance to the pricing problems than an extreme dual solution, although it is also argued that IPM are best suited to solving the large scale problems arising in decomposition [GSV97]. Goﬃn et al noted that the master problem need only be solved to within a certain tolerance of the optimal solution at each cycle, thus saving work in the IPM [GHVZ93]. An approximate warm start for the IPM method was applied. Brief experience with multi-commodity network ﬂow problems suggested that convergence of the decomposition method was always faster. Picking up the work of Goﬃn et al, Gondzio, Sarkissian and Vial used a new implementation of the analytic centre cutting plane method (ACCPM) [GSV97]. They tested the method on mostly random instances of nonlinear multi-commodity network ﬂow problems, with up to 1000 blocks, 2000 constraints and 80,000 variables in each block, and 4000 global constraints. No comparisons were given with other methods, but the authors themselves commented on the long solution times, attributed to having to solve the master problem from scratch on each cycle, with some instances taking 24 cycles to converge. Comparisons with a classical simplex based decomposition have been carried out by Chardiare and Lisser, with the simplex variant coming out a clear winner [CL96]. They questioned whether the belief in the superiority of interior points over extreme points in the context of decomposition is warranted. Gondzio andSarkissian used an implementation hopdm of an alternative IPM, the primal dual logarithmic barrier method (PDCGM) [GS96]. Following the earlier idea of solving the master problem to a certain tolerance, they suggested gradually increasing the tolerance so that less work is done in the early stages of the algorithm, but greater precision is obtained closer

THE USE OF INTERIOR POINT METHODS

to the optimal solution. Other advanced features of the implementation included a form of warm start and a removal of redundant proposals from the master problem—‘essential’ if the Cholesky factorisation is to remain sparse. Tests showed the purge strategy reduces CPU time by 50%. Using the same test set as [GSV97], they found that their hopdm implementation was on average 25% faster than the accpm implementation on all but the smallest problems, but that classical simplex based decomposition was faster on all but the largest problems, where the simplex variant fails completely. Their implementation of the simplex variant was missing many advanced features—such as the purge—that gave improved performance in previous work. Martinson and Tind used IPM to solve both the master problem and the pricing problems within the decomposition method [MT99]. By using interior points on the central path between the analytic centre and the optimal point, they viewed classical simplex based decomposition and the analytic centre IPM variant of Goﬃn et al as special cases. They claimed fewer cycles are required in general than either of these two cases, but based this claim on a test set of random small dense problems. The classical simplex based decomposition did win on several instances, and the authors’ implementation of simplex based decomposition was poor, with no multiple proposal generation for instance, which raises more doubts about the conclusions. Indeed the authors themselves admitted that little attention was paid to the computational eﬃciency of any of the implementations, as they just wanted a uniﬁed platform for comparative testing. Gondzio and Vial suggested and evaluated three enhancements to IPM based decomposition [GV99]. They used a trust region around the dual variables (prices) of the master problem, a form of warm start that speeded up solution of the restricted master and prevented severe ﬂuctuations in the prices, apparently with no negative eﬀects. They halted the pricing problems early with guaranteed ε-optimality, which saved IPM iterations when solving the pricing problems but may also have lead to an increased number of algorithm cycles. This technique, also suggested in [MT99], can be seen as a form of suboptimal proposal generation as implemented by, e.g., [BHS65], but with a guarantee on the optimality gap. They implemented an approximate warm start, by recording a weaker ε0-optimality solution (ε0 > ε) to be used on the next cycle to restart the pricing problem. The looser tolerance was necessary so that the solution is a valid IPM solution with the updated prices (and recall that the change in the prices is bounded by the trust region). The resulting implementation, developed from the accpm software for the master problem and hopdm for the pricing problems, can properly be classiﬁed as the ﬁrst second generation IPM based decomposition code. The three enhancements were tested using fmy problems from real applications, with between two and 16 pricing problems, and up to 42,000 constraints and 34,000 variables in total. The trust region resulted in sav

SURVEY OF COMPUTATIONAL EXPERIENCE

ings of between 17–66%; the suboptimal proposal generation a saving of about one ﬁfth, and the warm start a saving of about one third. But only the numbers of iterations were reported, not CPU time, which makes it diﬃcult to assess their work. Gondzio, Sarkissian and Vial used the software of [GV99] as a basis for a straightforward parallel implementation of IPM based decomposition [GSV01]. They used a set of 16 Linux Pentium Pro PCs linked by 100Mb/s ethernet and an implementation (mpich) of the MPI message passing standard. Using the same test set as [GSV01], they obtained reasonable speed-ups of 1.54–1.94 with two processors; 3 with fmy processors; and 5 with eight processors. The chief advantage of IPM based decomposition would appear to be improved numerical stability for degenerate problems, and given the reported numerical problems with decomposition, this may prove important. The main drawback is still the extra computational eﬀort required to solve the master problem and pricing problems repeatedly us