T.C.
BALIKESİR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ENDÜSTRİ MÜHENDİSLİĞİ ANABİLİM DALI

GEZGİN SATICI ÖRNEK PROBLEMLERİNİN OPTİMUM SONUÇLARININ GRİD ARACILIĞI İLE HESAPLANMASI

YÜKSEK LİSANS TEZİ

Mustafa ÇETİN
Bilgisayar Mühendisi

Balıkesir, Nisan - 2007

T.C.
BALIKESİR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ENDÜSTRİ MÜHENDİSLİĞİ ANABİLİM DALI

GEZGİN SATICI ÖRNEK PROBLEMLERİNİN OPTİMUM SONUÇLARININ GRİD ARACILIĞI İLE HESAPLANMASI

YÜKSEK LİSANS TEZİ

Mustafa ÇETİN

Tez Danışmanı : Doç.Dr. Ramazan YAMAN

Sınav Tarihi: 27/ 04 / 2007

Tez Danışmanı : Doç.Dr. Ramazan YAMAN (BAÜ Endüstri Mühendisliği)

Jüri Üyeleri : Yrd.Doç.Dr. Davut AKDAŞ (BAÜ Elektrik Elektronik Mühendisliği)

Yrd.Doç.Dr. M.Kubilay EKER (BAÜ Elektrik Elektronik Mühendisliği)

Balıkesir, Nisan – 2007

ÖZET
GEZGİN SATICI ÖRNEK PROBLEMLERİNİN OPTİMUM SONUÇLARININ GRİD ARACILIĞI İLE HESAPLANMASI

Mustafa ÇETİN
Balıkesir Üniversitesi, Fen Bilimleri Enstitüsü
Endüstri Mühendisliği Anabilim Dalı

(Yüksek Lisans Tezi / Tez Danışmanı : Doç.Dr.Ramazan YAMAN)

Balıkesir, 2007

İnsanoğlu günlük hayatta pek çok problemle karşılaşmaktadır. Bu problemleri aşabilmek için değişik çözüm yöntemleri kullanmaktadır. Bir problem matematiksel model ile ifade edildiğinde çözüme bir adım daha yaklaşılmış olunmaktadır. Bu şekilde birçok problem, bilgisayar teknolojisi kullanılarak çözülebilir hale gelmektedir.

Bazı günlük hayat problemlerinin çözülmesi zor, hatta bazen imkansızdır. Bunun sebebi problemlerin gerektirdiği işlem gücü ve zaman unsurlarının büyüklüğüdür. Bu gereksinimin üstesinden gelmek için paralel hesaplama teknolojileri kullanılmaktadır.

Bu çalışmada çözülmesi zor olan problemlerden biri olan Gezgin Satıcı Problemi (Travelling Salesman Problem) ele alınmış, sezgisel yöntemlere yol göstermesi amacıyla en iyi sonuçların elde edilebilmesi için çözüm yöntemi olarak Kaba Kuvvet (Brute Force) metodu kullanılmıştır.

ANAHTAR SÖZCÜKLER: Kaba Kuvvet / Paralel Hesaplama / Kombinatoriyel Optimizasyon / Gezgin Satıcı Problemi / Grid

ABSTRACT

CALCULATION OF OPTIMUM RESULTS TO THE TRAVELLING SALESMAN PROBLEM BY WAY OF GRID

Mustafa ÇETİN

Balikesir University, Institute of Science, Department of Industrial Engineering

(M.Sc. Thesis / Supervisor: Assoc. Prof. Dr. Ramazan YAMAN)

Balikesir - Turkey, 2007

Human being faces so many problems in his daily life. One uses different ways to tackle these problems. When a problem is expressed in a mathematical terminology, we are one further step closer to the solution. Thus, it enables us to solve many problems by using computer technology.

Some of the daily problems are hard and, sometimes, even impossible to solve. It is because of the magnitude of processing and time required by the problems. To cope with this requirement, parallel computing technologies are used.

In this work, we looked into Travelling Salesman Problem, one of the hardest problems to solve, and used Brute Force as the solution method to get the best results and thus show the way to Heuristic Algorithms.

KEY WORDS : Brute Force / Parallel Processing / Combinatorial Optimization / Travelling Salesman Problem / Grid

İÇİNDEKİLER

KISALTMALAR LİSTESİ

Adı	Tanımı
GSP	Gezgin Satıcı Problemi
P	Polinomial
NP	Non-Polinomial
WAN	Wide Area Network (Geniş Alan Ağı)
SISD	Single Instruction Single Data (Tek Komut Tek Veri)
MISD	Multiple Instruction Single Data (Çok Komut Tek Veri)
SIMD	Single Instruction Multiple Data (Tek Komut Çok Veri)
MIMD	Multiple Instruction Multiple Data (Çok Komut Çok Veri)
PVM	Parallel Virtual Machine (Paralel Sanal Makine)
MPI	Message Passing Interface (Mesaj Gönderme Arayüzü)
AMD	AMD firmasının ürettiği İşlemci markası
Intel Xeon	İntel firmasının ürettiği İşlemci markası
OPT.	Optimum
OPT.UZ.	Optimum uzunluk
sn	Saniye
sa	Saat
dk	Dakika

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Gezgin satıcı problemi çok bilinen, birçok problemin temelini teşkil eden iskelet bir problemdir. Bu problemin kesin sonuçlarının elde edilebilmesi, diğer problemlerin çözümüne de ışık tutmaktadır.

Bu çalışmada Optimizasyonun tanımı ve mevcut sınıflaması incelenmiş, ele aldığımız problemin de içinde olduğu Kombinatoriyel Optimizasyon problemleri açıklanmış, kesin sonuçlarının bulunabilmesi için kullanılan kaba yönteme değinilmiştir. Kaba yöntemdeki büyük işlem gücü ve zaman gereksinimine çare bulabilmek için paralel hesaplama teknolojisi irdelenmiş, örnek problemlerle hem tek bilgisayarda, hem de aynı anda farklı bilgisayarlara çözdürülerek aynı sonuçlar elde edilmiştir.

Bu çalışmanın her aşamasında sağladığı destekle beni yönlendiren değerli hocam Doç. Dr. Ramazan YAMAN'a gösterdiği ilgi ve sabır dolayısıyla teşekkürü bir borç bilirim.

Beni her zaman her konuda destekleyen pek değerli biricik aileme teşekkürlerimi sunarım.

Balıkesir, 2007 Mustafa ÇETİN

1. GİRİŞ

İnsanoğlu günlük hayatta pek çok problemle karşılaşmaktadır. Bu problemleri aşabilmek için değişik çözüm yöntemleri kullanmaktadır.

Problem matematiksel model ile ifade edildiğinde çözüme bir adım daha yaklaşılmış olunur.

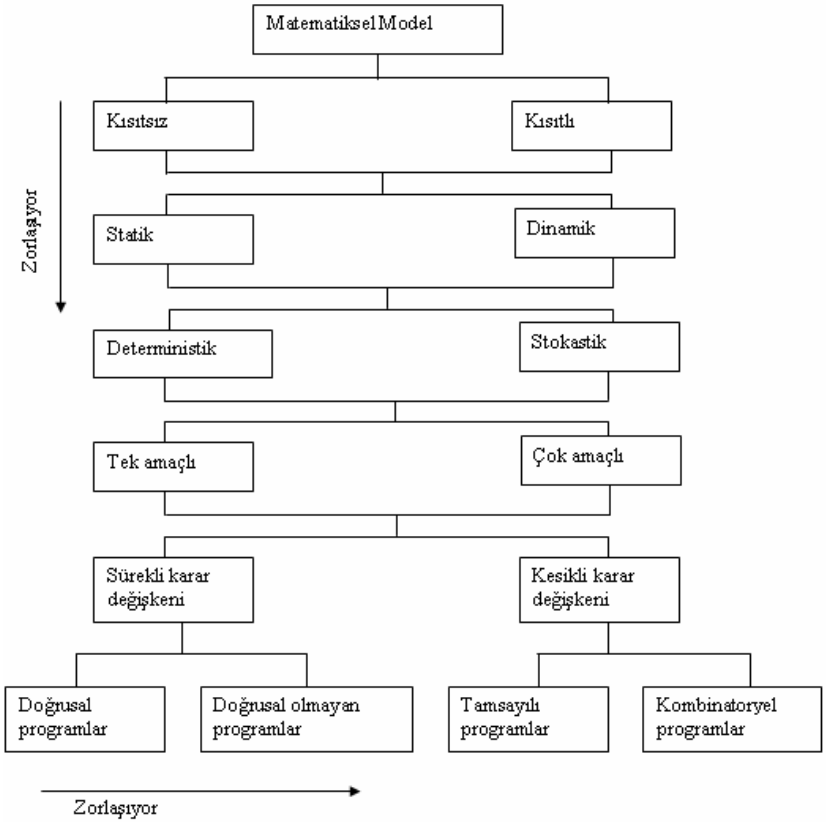
Problemleri çözerken kurulan matematiksel modeller kullanılan elemanları üç ana grupta toplamak mümkündür:

- amaç fonksiyonu,
- karar değişkenleri,
- kısıtlar.

Bir karar verme durumunda ilgilenilen sistem dikkatli bir şekilde gözlemlenir, değerleri kontrol edilebilen parametreler ve sistemin performansını etkileyen unsurlar belirlenir. Bu parametreler yöneticilerin kontrolü altındadır ve karar değişkenleri olarak tanımlanırlar. Bir üretim sisteminde farklı ürünlerin üretilecek miktarları, bir yerden başka yere taşınacak ürün miktarı, işçi sayısı, makine sayısı vb. unsurlar karar değişkeni olabilir.

Karar değişkenlerinin amaç üzerindeki etkilerinin analitik olarak gösterilmesiyle amaç fonksiyon oluşturulur.

Kısıtlar ise, sistemin içinde bulunduğu koşullardan kaynaklanmaktadır. Bunlar talep kısıtları, kapasite kısıtları vb. unsurlar olabilir.



Şekil 1.1 Problemin yapısına göre matematiksel model türleri [1]

Eğer karar değişkenleri üzerinde hiçbir sınırlama yoksa kısıtsız modeller ortaya çıkar. Gerçek hayatta karşılaştığımız modellerden bazıları bu yapıdadır. En az bir sınırlama olması kısıtlı modelleri ortaya çıkarır. Eğer problem tek bir dönem için çözülecekse statik model, birden fazla dönem göz önüne alınarak çözülecekse dinamik model ortaya çıkar. Eğer birden fazla amaç varsa çok amaçlı problemler ortaya çıkar. Eğer tüm karar değişkenleri pozitif reel değerler alıyorsa sürekli optimizasyon problemi söz konusudur. Tüm karar değişkenleri tamsayı değerler aldığında kesikli optimizasyon problemi ortaya çıkar. Bazı karar değişkenlerinin reel, bazılarının tamsayı değer alması durumunda ise karışık kesikli optimizasyon problemi ile karşılaşırız. Eğer karar değişkenlerinin kombinatoriyel seçenekleri söz konusuysa kombinatoriyel optimizasyon problemleri ortaya çıkar.

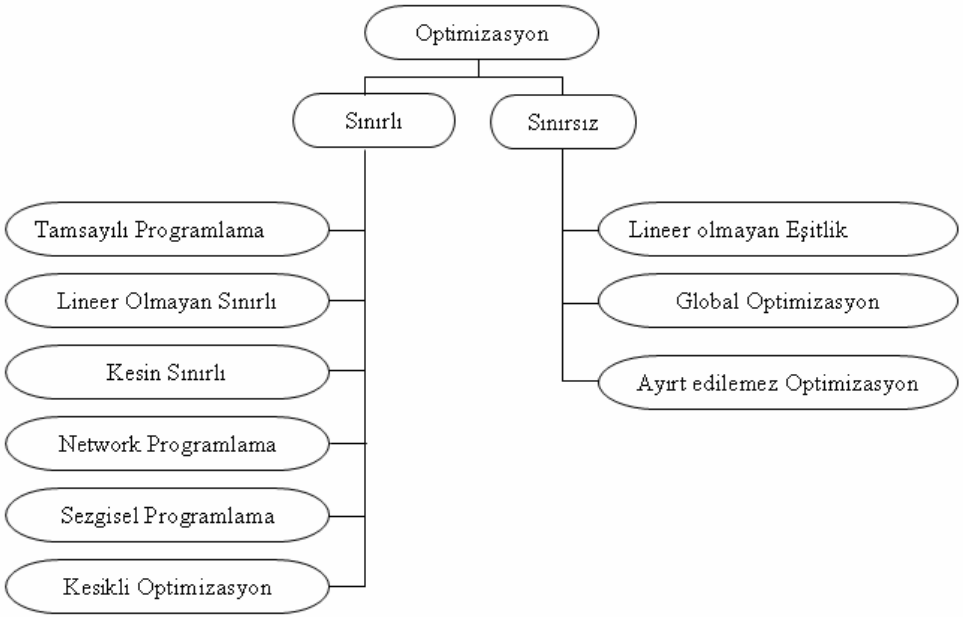
Dinamik modeller için kullanılan yaklaşım dinamik programlamadır. Modeldeki tüm fonksiyonların doğrusal olması durumunda sürekli optimizasyon problemleri doğrusal programlama yöntemi ile çözülür. Sürekli optimizasyon

modelinde en azından bir fonksiyonun doğrusal olmaması durumundaysa doğrusal olmayan programlama yöntemi kullanılır. Eğer kesikli optimizasyon problemlerinde karar değişkenleri herhangi bir tamsayı değer alıyorsa tamsayılı programlama yöntemi kullanılır [1].

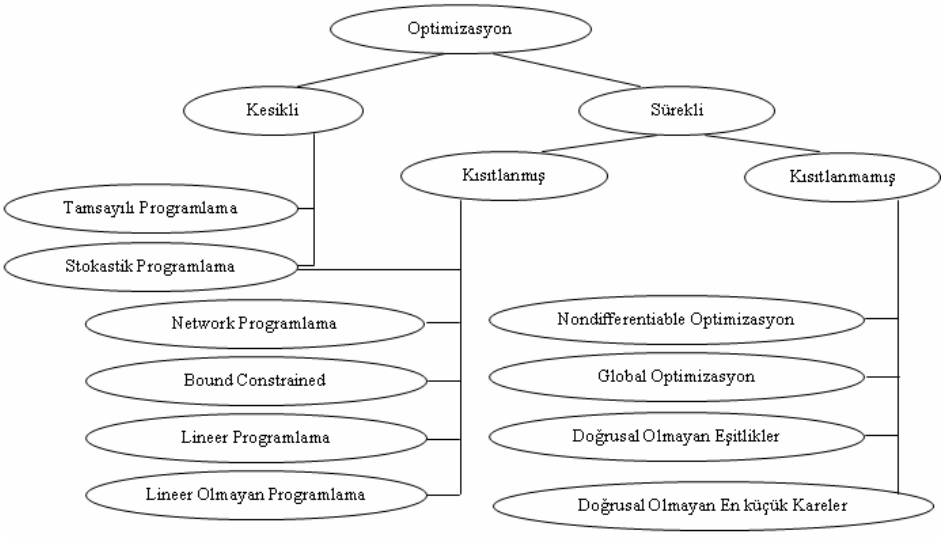
2. OPTİMİZASYON

Bir problemin en iyi çözümünün bulunabilmesi için mevcut çözüm alternatifleri içinden seçim yapmak ve buna karar vermektir. Diğer bir ifade ile problemin sonucunun, ulaşılması gereken amaca en uygun olduğu değeri bulma işlemidir. Kısacası en iyi sonuca ulaşmaktır.

2.1 Optimizasyon problemlerinin sınıflandırılması



Şekil 2.1 Optimizasyon Problemleri [2]



Şekil 2.2 Optimizasyon Problemlerinde NEOS Sınıflaması [3]

Şekil 2.2 de gösterildiği gibi Optimizasyon problemleri, karar değişkenlerinin sürekli veya kesikli olmasına göre ikiye ayrılmaktadır. Birinci bölümde de bahsedildiği üzere, bir optimizasyon probleminin değişkenleri tamsayı değerlerden oluşuyorsa, bu problemi kesikli optimizasyon problemi sınıfına dahil ederiz. Bu tip problemler, kombinatoriyel optimizasyon problemleridir.

Kombinatoriyel optimizasyon problemlerinin belirli bir boyuta kadar olanı tamsayılı programlama yöntemi ile çözülürken, orta ve büyük boyutlu problemlerin sezgisel yöntemlerle çözülmesi gerekmektedir [1].

2.1.1 Kombinatoriyel Optimizasyon

Karar değişkenlerinin kesikli (süreksiz) olduğu optimizasyon problemlerine kombinatoriyel optimizasyon problemleri denir [4].

Mümkün çözümler kümesinin kesikli olduğu veya kesikli olana azaltılabildiği, amacın en iyi çözümün bulunması olduğu bir optimizasyondur [5].

2.1.2 Örnek Kombinatoriyel Optimizasyon Problemleri

- Atama Problemi (The Assignment Problem)
- Sırt Çantası Problemi (0-1 Knapsack Problem)
- Küme Kapsama Problemi (Set Covering)
- Araç Rotalama Problemi (Vehicle Routing)
- Gezgin Satıcı Problemi (Travelling Salesman Problem)
- Ağ ve Graf Problemleri (Network and Graph)

2.1.2.1 Atama Problemi (The Assignment Problem)

n elemanın, n farklı göreve atanması problemidir. i. kişinin, j. işi yapma maliyeti c_{ij} dir. .Problem,

$$\sum_{i=1}^{n} c_i \prod_i \tag{2.1}$$

$$\{\prod_1, \ldots\ldots, \prod_n\} \tag{2.2}$$

(2.1) fonksiyonunu minimize edecek (2.2) atama kümesinin bulunması şeklinde tanımlanabilir. Burada problem çözümü,

$$\{1, \ldots, n\} \tag{2.3}$$

$$\{\prod_1, \ldots\ldots, \prod_n\} \tag{2.4}$$

(2.3) sayılarının (2.4) permütasyonu olarak gösterilmektedir [6].

2.1.2.2 Sırt çantası problemi (0-1 Knapsack Problem)

n parçanın, kapasitesi C birim olan bir sırt çantasına yerleştirilmesi problemidir. i. parça v_i değerine sahip olup kapasitenin c_i birimini kullanmaktadır. Buna göre I altkümesinin belirlenmesi istenmektedir.

$$c_i = \sum_{i \in I} v_i \tag{2.5}$$

$$\sum_{i \in I} c_i \leq C \tag{2.6}$$

Ekonomik ve teknik sistemlerdeki birçok karar verme problemleri de Knapsack tipli problemler şeklinde gösterilebilir.

I={1,2,...,m} ve J={1,2,...,n} olmak üzere, b_i , c_j pozitif tam sayıları ve negatif olmayan a_{ij}, i∈I , j∈J sayıları verilmiştir.

$$R_m = \max_x \left\{ \sum_{j \in J} c_j x_j \Big| \sum_{j \in J} a_{ij} x_j \leq b_i, i \in I, x_j = 0 \vee 1, j \in J \right\} \tag{2.7}$$

R_m fonksiyonunun bulunması istenmektedir.

Bu problem çeşitli açılardan incelenmiş ve çeşitli yönlerde genelleştirilmiştir. Sürekli olarak problemin yeni uygulama alanları ve yeni anlamlı yorumları ortaya çıkmaktadır. Bunun en yaygın yorumu aşağıdaki gibi verilebilir.

P_1, P_2,...,P_n gibi n tane proje ve her P_j projesinden beklenilen kazanç c_j, bu projenin gerçekleşmesi durumunda K_i kaynağına olan ihtiyacı a_{ij} , ayrılmış olan K_i kaynağının kapasitesi b_i ile verilmiştir.

$$x_j = \begin{cases} 1, & \text{eğer } P_j \text{ projesi gerçekleşiyorsa} \\ 0, & \text{aksi halde} \end{cases}$$

olmak üzere x_j (j∈J) değişkenleri alınarak R_m problemi elde edilir.

Bu problem, verilen kaynak hacmini arttırmamak koşuluyla, toplam gelir maksimum olacak şekilde, verilen projeler kümesinden bir altküme seçilmesi olarak yorumlanabilir.

Sırt çantası problemi, özellikle ekonomi alanında rastlanan bir tamsayı problemidir. Uygulamada karşılaşılan problemler yüzlerce sınırlayıcı koşuldan ve binlerce karar değişkeninden oluşmaktadır [7].

2.1.2.3 Küme Kapsama Problemi (Set Covering Problem)

Tüm kısıtların $\geq$ eşitsizliğine sahip olan, tüm sağ taraf değerlerinin 1 olduğu ve katsayı matrisinin 0-1 matrisi olan saf 0-1 tamsayılı programdır.

Şöyle ifade edilebilir :

$$\min \ z = \sum_{j=1}^{n} c_j x_j \tag{2.8}$$

Kısıtlar $\quad A\,x \geq b\,, \quad x_j = 0 \ veya \ 1 \quad j=1,2,...,n \tag{2.9}$

Burada A, (m x n) şeklinde kısıt kat sayılarından oluşan bir 0-1 matris; b ise değeri 1 olan bir sağ taraf vektörüdür. Amaç fonksiyonu $c_1x_1 + c_2x_2 + ... + c_nx_n$ değerini sıfır yapacak şekildedir. Her $j=1,2,...,n$ 'ler için c_j , j yerindeki tesis maliyetini gösteriyorsa bu durumda katsayıların 1'den başka değerler alabileceği de varsayılır.

Bu matematiksel modelde kısıtların işaretine ve sağ taraf değerine bağlı olarak özel problem tipleri ortaya çıkmaktadır. Eğer kısıtlardaki eşitsizlik, eşitlik haline getirilirse ve sağ taraf değeri olan b, 1 olarak kalırsa, bu tip probleme *küme bölünme problemi* denir. Eğer kısıtların hepsi ($\leq$) hâline getirilirse ve sağ taraf değeri *b, 1* olarak kalırsa, *küme paketleme problemi* denir. Dağıtım ve rotalama problemleri, planlama problemleri ve yerleştirme problemleri gibi problemler sık sık küme kapsama yapısında karşımıza çıkmaktadır [4].

2.1.2.4 Araç Rotalama Problemi (Vehicle Routing)

Bu problemde, verilen amaç fonksiyonuna göre araçlara rotalar belirlenir. Güzergah belli bir depodan başlar ve coğrafik olarak dağılmış müşterilere tek bir araçla bir kez uğranarak talepleri karşılanıp tekrar depoya dönülür. Gezen aracın müşterilerin ihtiyaçlarını karşılayabilecek kapasitede olması gerekir. Yani toplam

müşteri talebi, araç kapasitesine eşit veya daha az olmalıdır. Bu durumda tek bir araç, bir defada tüm müşterilere uğrayarak taleplere cevap verebilir.

Araç kapasitesinin sınırsız olduğu durumlarda, depo veya müşterilerin arz ve talepleri yüzünden, Araç Rotalama Problemi, Gezgin Satıcı Problemine (GSP) dönüşür [4].

Bir depo, n müşteriye dağıtım yapmak üzere m tane araca sahiptir. i. müşteri c_i birime ihtiyaç duymaktadır. k. aracın kapasitesi c_k 'dır. i. ve j. müşteriler arasındaki uzaklık $d_{i,j}$ olsun. Hiçbir araç D birimden fazla taşıyamamaktadır. Bu problemde müşterilerin araçlara atanması ve her bir aracın müşterilerini ziyaret etme sırasının bulunması istenmektedir:

Burada, k aracı n_k müşteriyi ziyaret etmektedir ve problem çözümleri, n_k sayılarına bölüştürülen $\{1, ..., n\}$ sayılarının permütasyonu olarak gösterilmektedir.

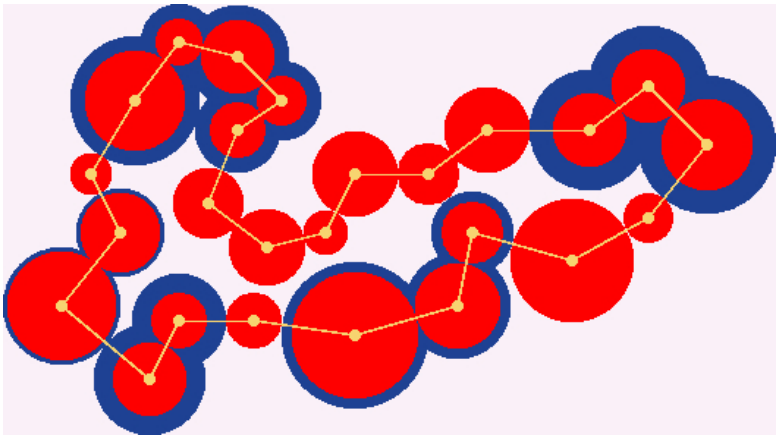
Bu problem birçok ticari uygulamada karşımıza çıkmaktadır. Bu nedenle çok sayıda araştırmacının ilgisini çekmiştir ve 60'lı yıllardan bu yana üzerinde yoğun biçimde çalışılmaktadır. Çok sayıda sezgisel veya kesin yaklaşım önerilmiştir [8].

2.1.2.5 Gezgin Satıcı Problemi (Travelling Salesman Problem)

Şehir kümesi ve her iki şehir arası uzaklıklar verilmiş olsun. Gezgin satıcı problemi (GSP), tüm şehirleri ziyaret ederek başlangıç noktasına dönüldüğünde en ucuz yolu bulmaktır.

GSP problemi en dikkat çekici kombinatoriyel optimizasyon problemlerindendir. Tanımlaması basit, simülasyonunun zorluğu ile ün salmış ve hala etkili algoritmalar bulunmaya çalışılan bir problemdir [10].

Bir satıcının n adet şehri (veya düğümü) dairesel olarak (yani işi bittiğinde başladığı noktaya geri gelecek şekilde) gezerken minimum mesafe kat etmesidir.

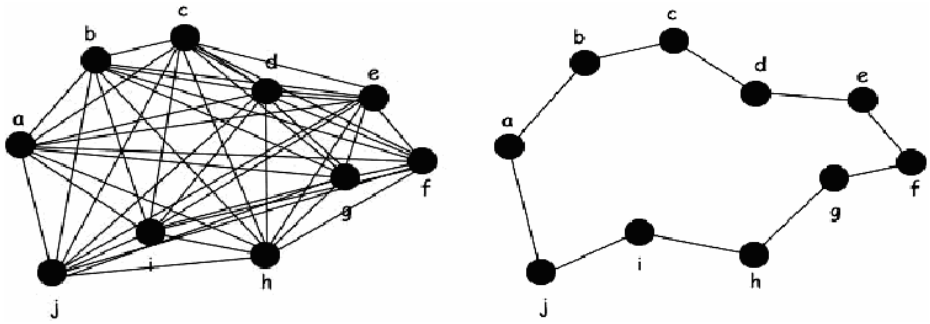


Şekil 2.3 Gezgin Satıcı Problemi [9]

Şekil 2.3'deki noktalar şehirleri temsil etmektedir. Bu şehirleri gezerken tüm şehirlere uğramak zorunda olduğu gibi bir şehirden de sadece bir kere geçebilir [11].

GSP, sonlu sayıda verilen şehir adedi ve her ikili arası uzaklık verildiğinde bir noktadan başlayarak, başlama noktasına dönülmek üzere tüm şehirleri dolaşarak en ucuz yolun bulunmasıdır [12].

GSP problemi, bir kişinin verilen şehirler üzerinden sadece bir kez geçmek koşuluyla şehirleri dolaşarak en kısa turu bulması problemidir [13].



Şekil 2.4 Gezgin Satıcı Problemi [13]

Bir satıcı bir iş gününü, bir şehirden başlayarak *n* adet şehri döngüsel olarak ziyaret ederek geçirecektir. Burada problem sıralı gezi turunu *n* adet şehir için, her şehri bir defa ziyaret etmek koşuluyla minimize etmektir [14].

GSP ilk bakışta basit gibi görünen kombinasyonel bir problemdir. Bir satıcı *n* adet şehri (veya noktayı) dairesel olarak (yani işi bittiğinde başladığı noktaya geri gelecek şekilde) gezerken minimum mesafe kat etmelidir. Bu şehirleri gezerken tüm şehirlere uğramak zorunda olduğu gibi bir şehirden de sadece bir kere geçebilir [11].

Gezgin satıcı problemi şu şekildedir:

Bir gezgin satıcı mallarını *n* şehirde satmak istiyor, Fakat mantıklı bir şekilde, bu şehirleri mümkün olan en kısa şekilde turlamak istiyor.

Problemin amacı, satıcıya bu en kısa yolu sunabilmektir.

- Satıcının ilk şehirde, *n* değişik şehir arasında seçim hakkı vardır,
- İkinci şehirde, *n-1* değişik şehir arasında seçim hakkı vardır,
- Bu şekilde devam edildiğinde *n.*şehirde bir adet şehir (*n.* olanı) kalmıştır.
- Dolayısıyla, sonuç olarak satıcının *n!* değişik tur arasından seçim hakkı olacaktır. Bu, 100 şehirlik bir tur için bile $9.33 * 10^{157}$ değişik tur etmektedir [15].

Gezgin satıcı problemi ulaşılacak olan düğümler arası uzaklığın bilindiği ve her birisinden yalnızca bir kez geçilerek en kısa yolun (maliyetin) bulunduğu tam sayılı programlama yöntemidir.

Gezgin satıcı problemi modeli kurulan sistemdeki istenen parçaları veya nesneleri en kısa zamanda, en az maliyet ve en çok kar ile toplamak, dağıtmak gibi tanımlayabiliriz [16].

Gezgin satıcı problemi, kesikli optimizasyon problemlerinden birisidir. Müşteriler, işler, şehirler, noktalar vb. için belli bir güzergah ya da ardışıklık tayin etme problemidir. Matematiksel programlama literatüründe tamsayılı programlamanın en çok üzerinde çalışılan konularından birisidir.

Nesnelerin en küçük toplam maliyet, uzaklık veya zaman alacak şekilde sıralanması gezgin satıcı problemlerine girmektedir.

Örnek gezgin satıcı problemleri:

- Anakart üretiminde parçaların anakarta yerleştirme sırasının belirlenmesinde anakartı üretmek için gerekli zamanı enküçüklemede parçaların yerleştirileceği yerler arasındaki uzaklıklar dikkate alınır.
- Ana depodan dağıtımı gerçekleştirecek bir kamyonun güzergah tayini de GSP dir.
- Sıraya bağımlı kuruluş zamanlarının (setup time) en küçüklenmesi istendiği üretim süreci de bir GSP dir. Birden fazla iş, bir makinede işlem göreceği zaman her bir iş için toplam kuruluş zamanı genellikle hangi işin hangi sırada işlem görmesine bağımlıdır. Bunun için optimal işlem sıralarının belirlenmesi gerekmektedir [17].

Gezgin satıcı problemi statüsünde çalışılmış örnekler:

- Lenstra ve Rinnooy Kan yaptıkları çalışma ile bilgisayar anakartındaki pinlerin kablolarla birleştrilmesi problemi üzerinde çalışmışlar, toplam kullanılacak kablo uzunluğunu en küçükleyecek model kurmuşlardır.
- Garfinkel, rulo halindeki duvar kağıtlarını atıkları en küçükleyecek şekilde n parçaya ayırma problemini GSP olarak modelleştirmiştir. Amacı toplam atıkları en küçükleyecek şekilde *n* kağıdın sırasını belirlemek olarak ele alınmıştır.
- Reinelt metal levhaların üzerinde delik açma problemini ele almış. Delgi makinesinin delikleri açmak için toplam kat edeceği mesafeyi en küçüklemek istemiştir.

İmalat sistemlerinde iş ardışıklığı problemi de GSP olarak ele alınabilmektedir. *n* tane işin tek bir makinede işlem göreceği varsayılırsa böyle bir problemde *i* işinden sonra *j* işinin başlatılması için arada sistemde değişiklik yapabilmek için harcanacak zaman söz konusudur. Bu problemde amaç *n* adet iş için harcanan toplam zamanı en küçükleyecek işlem ardışıklığının bulunmasıdır [17].

2.1.2.5.1 Gezgin Satıcı Probleminin Kullanım Alanları

- GSM operatörlerinin baz istasyonlarının yerleşim yerlerinin belirlenmesi problemi,
- Birçok ulaşım ve lojistik uygulamaları,
- Malzeme akış sistem tasarımı,
- Araç Rotalama Problemleri,
- Depolardaki vinç güzergahlarının programlaması,
- Stok alanındaki malzeme toplama problemleri,
- Uçaklar için havaalanı rotalaması,
- Elektronik devre tasarımı [18].

Teknoloji, üretim, ticari ve bunun gibi birçok alanda, kısacası günlük hayatta ihtiyaç duyduğumuz birçok işlemin temel mantığını teşkil eder.

2.1.2.5.2 Problemin Yapısı

Gezgin satıcı problemi ikiden fazla şehir arasında, her şehre bir kez uğramak koşuluyla en kısa yolu kat edebilecek şekilde dolaşılacak rotanın bulunması mantığı ile oluşturulur.

Her şehre bir kez uğramak koşulu ile denenebilecek bütün ihtimallerden oluşan şehir turları arasındaki en kısa mesafeli rotaya ulaşılmak istenmektedir. Bunun için ilk denenen rota en kısa mesafeli olarak kabul edilir. Sırasıyla diğer rotaların uzunluklarıyla karşılaştırılır. Sonuçta en küçük uzunluğa sahip rota optimum tur olarak belirlenir.

Şehir sayısı arttığında işlem karmaşıklığı üstel olarak artmaktadır. Problem hesaplama karmaşıklığı *(şehirsayısı)!* olarak gösterilebilir. Örnek olarak, 6 şehir için 6! ihtimal, 7 şehir için 7! ihtimal vb. gibi.

2.1.2.5.3 Problemin Çözülemezliği

Çözülmesi zor olan kombinatoriyel optimizasyon problemlerinde, çözüm uzayının tamamının taranmasını gerektiren geleneksel çözüm yöntemlerinde problem çözümü değişken sayısının artması ile imkansız hale gelebilmektedir.

Bu nedenle problemlerin kısa zamanda değişik yaklaşımlarla çözümü düşünülmüş, çözüm uzayının yalnızca belirli bir kısmının taradığı sezgisel yöntemler denenmiştir. Çözüm uzayının tümü gözden geçirilmediği için sezgisel yöntemlerle gerçek optimum sonuca ulaşma garantisi yoktur.

Bilindiği gibi gezgin satıcı problemi de kesikli bir kombinatoriyel optimizasyon problemidir. Gezgin satıcı problemindeki gezinilen her nokta birbiri ile ilişkilidir. Herhangi bir noktanın yerinin değişmesi, sonucu etkilemekte, problemin çözümü sonucu bulunan optimum rota değişebilmektedir. Bu nedenle noktalardan herbirinin yeri problemin sonucu açısından önemlidir. Bir noktanın koordinatının değişmesi problemin optimum değerini değiştirmektedir.

Bu nedenle problem, kısmen parçalansa bile, tamamen parçalanamaz bir yapıya sahiptir. Mevcut probleme eklenebilecek her bir nokta problemin hesaplama karmaşasını *eklenmeden önceki nokta sayısı+1* kere daha güçleştirmektedir. İlk aşamada çözülebilecek gibi görülen GSP problemi, belirli noktadan sonra çok uzun zaman gereksinimi duymakta, gittikçe çözülemez bir hale gelmektedir.

2.1.2.5.4 Polinomsal İfadesi

Şehirler kümesi $i=\{1,..,n\}$ olmak üzere, şehir kümesindeki elemanlar yalnız birer kez kullanılarak yerleşim olasılıkları belirlenir.

j, n adet şehrin herbirisi bir kez kullanılarak oluşturulabilecek *n!* farklı ihtimalden birisidir.

k, herhangi bir *j* ihtimali için hesaplanan şehir yerleşiminde bulunan bir şehri göstersin.

$[a_{jk}]_{n!\times n}$, elemanları $a_{jk} \in \{1,2,..,n\}$ şehir kümesinden oluşan ve hesaplanacak *n!* rotada bulunan şehirlerin her birisinin yerleştirileceği matris olsun.

Herhangi bir p rotasında bulunan *q*. eleman için (a_{pq});

$x_{a_{pq}}$, koordinat eksenindeki apsisi,

$y_{a_{pq}}$, koordinat eksenindeki ordinatını göstersin.

$$g(p)=\sum_{q=1}^{n}\sqrt{(x_{a_{pq}}-x_{a_{p(q-1)}})^2+(y_{a_{pq}}-y_{a_{p(q-1)}})^2}\,, \quad p\in\{1,2,..,(n-1)!,n!\} \qquad (2.10)$$

(2.10) ile ifade edildiği gibi, herhangi bir *p* rotasının uzunluğu *g(p)* fonksiyonu ile bulunur.

Problem için amaç fonksiyonu, her bir $p \in \{1,..,n!\}$ farklı ihtimal için şehir ikililerinin birbirine olan uzaklıklarının toplamını veren *g(p)* fonksiyon değerinin minimize edilmesidir.

2.1.2.5.5 P ve NP Sınıfı Problemler

Hesaplama teorisinde bazı problemlerin çalışma süresi, girilen verinin büyüklüğüne bir polinom cinsinden bağlı olduğu bilinmektedir. Bunlara polinomsal zamanda çalışan algoritma denilmektedir. Bu tip problemler, karmaşıklığı polinomsal olan **P** sınıfı problemlerdir.

Buna karşılık sorulan soru ve girilen verinin büyüklüğüne polinom mertebesinde bağımlı bir sürede çözüm algoritması bulunamayan problemler vardır. Bu tür problemlerin cevabı tahmin edilebiliyorsa tahminin doğruluğunu sınamak için veri büyüklüğüne polinom mertebesinde bağımlı sürelerde çalışabilecek algoritmalar vardır. İşte bu tip problemler **NP** (non-deterministic polinomial time) sınıfı problemleri oluşturur.

Karmaşık matematiksel hesapların belirli bir düzenek tarafından yapılıp yapılamayacağı 20.yy'ın başlarında büyük bir tartışma konusu olmuştu. Öteden beri el ile veya zihinden yapılan hesaplamalar çok zaman almakla birlikte, birçok hatayı da beraberinde getiriyordu. 1936 yılında, Alan M. Turing "Saptama Problemi Hakkında Bir Uygulamayla Birlikte Hesaplanabilir Sayılar" (On Computable Numbers, with an Application to the Entscheidungsproblem) isimli bir makalesini yayınladı. Makalesinde teorik ve matematiksel temellere dayalı sanal bir makineden bahseden Turing, her türlü matematiksel hesabın bu sanal makineyle yapılabileceğini iddia ediyordu. İşte bu sözü geçen sanal makine daha sonraları Turing Makinesi olarak isimlendirildi [19].

Belirli Turing Makinesi ile polinomsal zamanda doğrulanabilen her problem NP sınıfındandır. Belirli Turing Makinesi aynı zamanda Belirsiz Turing Makinesi olduğundan P sınıfı problemler aynı zamanda NP sınıfı problemlerdendir [20].

NP sınıfı P sınıfına çevrilmesine bir örnek verelim: Eğer bir sorunun cevabı, verinin büyüklüğüne polinom mertebesinde bağımlı sürede çalışacak bir algoritma ile bulunabiliyorsa, bu soruya bir cevap olarak üretilmiş bir tahminin doğruluğunu da verinin büyüklüğüne polinom mertebesinde bağımlı bir sürede çalışacak algoritma ile kontrol edebilmek mümkündür [21].

NP sınıfı problemleri, rasgele seçilen bir çözüm olasılığını test eden bir turing makinesi ile çözülebilen karar problemlerini içerirler. Yani NP sınıfı problemler, mümkün bütün çözümlerin denenmesi yöntemi ile çözülebilir. Buradan da, problemin çözümü için gerekli zamanın üstel fonksiyona bağlı olduğu anlaşılmaktadır [4]. NP sınıfı problemler üstel zaman karmaşıklık fonksiyonlarına sahiptir [22].

NP sınıfı, P sınıfını kapsamaktadır. Eğer bir problem için polinomsal zaman algoritması varsa üstel zaman alsın diye algoritma etkin olmayan biçimde üssel zaman algoritmasına dönüştürülebilir. Aynı zamanda gerçekte NP olan ancak P de olmayan bir problem için bir polinomsal zaman algoritması bulunabilirse P'ye taşınabilmektedir.

NP içerisinde P'ye asla taşınamayacak problemlerin bulunması, modern matematikçilerin ortaya koyduğu gerçeklerden biridir. Bir başka deyişle, birçok matematikçi, çok zor oldukları için bazı problemlere ait polinomsal zaman karmaşıklık fonksiyonuna sahip olan algoritmaların asla bulunamayacağına inanırlar. Çoğu çizelgeleme problemi, bu sınıf içerisinde yer alır ve NP-Zor (NP-Hard) olarak adlandırılır [22].

Hem NP, hem de NP-Zor olan problemler NP-Tam (NP-Complete) olarak adlandırılır. Bu tip problemler NP sınıfının en zor problemlerindendir. Herhangi biri çok terimli zamanda çözülebilirse bütün hepsi çok terimli zamanda çözülebilir.

NP-Tam Problemlere Örmekler [20]:

- İkilik tatmin edilebilirlik (CNF-SAT),
- Gezgin Satıcı Problemi (TSP),
- Hamilton Dönüşü ve Hamilton Yolu,
- Altküme Toplamı,
- Bağımsız Küme Problemi,
- Düğüm Kapsama Problemi.

Ele aldığımız GSP probleminin çözüm algoritması, *n* şehir sayısına bağlı olarak *O(n!)* karmaşıklığa sahiptir.

2.1.2.5.6 Problem Çözümü İçin Kullanılan Bazı Algoritmalar

Gezgin satıcı probleminin çözüm zamanı 15 şehre kadar kesin sonuçları kolayca bulunabilmektedir. Çözüm karmaşıklığı şehir sayısına üstel olarak bağımlı olduğu için 16 şehirden sonra imkansızlaşmaktadır. Bu nedenle çok şehirli GSP problemlerini çözebilmek için yaklaşık çözümler üreten sezgisel algoritmalar kullanılmaktadır.

Sezgisel algoritmalar belirli kriter ve yönergelere göre çözüme gitme yaklaşımını kullanır. Örneğin GSP için bir şehirden diğerine giderken henüz ziyaret etmediği şehirler arasında, en yakındakini tercih edebilir.

Almanya'nın 15112 adet şehrini gezen ve her şehirden yalnızca bir kez geçen en kısa yolu bulabilmek için Rice ve Princeton Üniversitelerindeki bilgisayarlar 22 yıldan daha fazla süre çalışarak yaklaşık 66000 km'lik rotayı bulmuşlardır [23].

2.1.2.5.6.1 Tabu Arama (Tabu Search)

Tabu arama algoritması, bir başlangıç seçimi ele alınarak bu seçimin komşuları belirlenir. Bir amaç fonksiyonu göz önüne alınarak komşular değerlendirilir. Bu değerlendirme esnasında arama listesi hafızada tutulur. Bu şekilde arama işlemi gittikçe sınırlanmaya başlar. Değerlendirilen komşular tabu arama listesinde yoksa ve aspirasyon kriterini sağlıyorsa, bulunmuş en iyi çözüm ile karşılaştırılır ve arama listesine eklenir. Son çözüm o ana kadar bulunmuş en iyi sonuç ise yeni başlangıç çözümü olarak seçilir. Bu işlem durma kriteri sağlanana kadar tekrar eder.

Tabu arama, mümkün bir çözüm ile başlar. Bu çözüm, problemin matematiksel ifadesinde geçen kısıtları tatmin eden bir çözümdür. Tabu aramanın performansı başlangıç çözümüne bağlıdır. Bu nedenle mümkün olduğunca iyi olan bir çözüm ile başlamak gerekir.

Tabu Listesi ilk giren ilk çıkar şeklinde tasarlanmış bir listedir. Kritere uyan çözümlerin karakteristik özellikleri tabu listesini sürekli olarak yukarıdan doldurmaya başlar. Bulunan elemanların sayısı liste uzunluğunu aştığında yeni gelen elemanın listeye eklenmesi için listenin en altındaki eleman silinir [24].

2.1.2.5.6.2 Genetik Algoritmalar

Algoritma ilk olarak popülasyon diye tabir edilen bir çözüm (kromozom) seti ile başlatılır. Bir popülasyondan alınan sonuçlar bir öncekinden daha iyi olacağı beklenen yeni bir popülasyon oluşturmak için kullanılır. Yeni popülasyon oluşturulması için seçilen çözümler uyumluluklarına göre seçilir. Çünkü uyumlu olanların daha iyi sonuçlar üretmesi olasıdır. Bu istenen çözüm sağlanıncaya kadar devam ettirilir [25].

Algoritma [26]:

1. Başlangıç: *n* adet kromozom içeren popülasyonun oluşturulması
2. Uyumluluk: her x kromozomu için uyumluluğun *f(x)* değerlendirilmesi,
3. Yeni popülasyon: Yeni popülasyon oluşuncaya kadar aşağıdaki adımların tekrar edilmesi,
    a. Seçim: Toplumdan uygunluklarına göre iki ata seçilir (daha iyi uyum, seçilme şansını artırır.)
    b. Çaprazlama: Yeni bir fert oluşturmak için ebeveynlerin bir çaprazlama olasılığına göre çaprazlanması. Eğer çaprazlama yapılmazsa yeni fert anne ve babanın aynısı olacaktır.
    c. Mutasyon: Yeni ferdin mutasyon olasılığına göre kromozom içindeki konumu değiştirilir.
    d. Ekleme: Yeni bireyin yeni popülasyona eklenmesi.
4. Değiştirme: Algoritmanın yeniden çalıştırılmasında oluşan yeni popülasyonun kullanılması,
5. Test: Eğer sonuç tatmin ediyorsa algoritmanın sona erdirilmesi ve son popülasyonun çözüm olarak sunulması.
6. Döngü: 2. adıma geri dönülmesi.

2.1.2.5.6.3 Brute Force

Tabu Arama ve Genetik Algoritmalar, kesin sonucu bulma garantisi olmayan yaklaşık çözümler saptayan yöntemlerdir. Brute force yöntemi ise bütün alternatifleri

deneyerek bunlar içinde en iyi sonucu arar.

Problemi tam olarak çözebilmek için olası tüm turların tutarı veya uzunluğu hesaplanıp bunların en iyisinin seçilmesi gerekmektedir. Tüm alternatiflerin denenmesine brute force veya geniş kapsamlı arama denir [10].

Brute force, gerçek optimum değerin/değerlerin bulunabilmesi için mevcut problemin çözüm uzayındaki her bir sonucun denenmesidir.

Problemimizde elimizdeki şehirlerin farklı kombinasyonları saptanır. Örneğin üç adet 0, 1, 2 şehirleri için; 3!= 6 farklı diziliş şekli vardır. Bunlar:1 2 0, 2 1 0, 2 0 1, 0 2 1, 1 0 2, 0 1 2 olarak gösterilebilir. Her bir olası diziliş için, diziliş içindeki birbirine komşu her şehir ikililerinin birbirine uzaklıkları toplamı ile toplam diziliş uzunluğu hesaplanır. Bütün olası alternatifler için bu hesaplama yapılarak sonuçta bulunan en kısa mesafeli diziliş, en iyi (optimum) sonuç olarak saptanmış olur. En optimum sonucu veren rota ise optimum rota olarak belirlenir.

2.1.2.5.6.3.1 Bilgisayar Algoritmaları

Brute Force metodu ile çalışan iki algoritma kullanılmıştır.

2.1.2.5.6.3.1.1 For Yapısı

```
for (a=0 ; a<= max ; a++)
 for (b=0 ; b<= max ; b++)
  for (c=0 ; c<= max ; c++)
   for (d=0 ; d <= max ;d++)
    for (e=0 ; e<= max ; e++)
     for (f=0 ; f <= max ;f++)
      for (g=0 ; g <= max ; g++)
       for (h=0 ; h <= max ; h++)
         for (i=0 ; I <= max ; i++)
          {
           if (
              (a!=b)&& (a!=c) && (a!=d) && (a!=e) && (a!=f) && (a!=g) && (a!=h) && (a!=i)&&
                       (b!=c) && (b!=d) && (b!=e) && (b!=f) && (b!=g) && (b!=h) && (b!=i)&&
                                 (c!=d) && (c!=e) && (c!=f) && (c!=g) && (c!=h) && (c!=i)&&
                                           (d!=e) && (d!=f) && (d!=g) && (d!=h) && (d!=i)&&
                                                     (e!=f) && (e!=g) && (e!=h) && (e!=i)&&
                                                               (f!=g) && (f!=h) && (f!=i)&&
                                                                         (g!=h) && (g!=i)&&
                                                                                   (h!=i)

              )
              {  printf ("\n%d %d %d %d %d %d %d %d", a,b,c,d,e,f,g,h,i);  }  // if sonu
         } // for sonu
```

Şekil 2.5 for yapısı ile kurulan algoritma

7 şehir için denendiğinde 5040 olasılık gözden geçirilmesi için 823.543 dönüm yapılmış. (1/20)

8 şehir için denendiğinde 40320 olasılık gözden geçirilmesi için 16.777.216 dönüm yapılmış (1/416) .

9 şehir için toplam denenmesi gereken olasılık adedi 362.880'dir. Fakat for yapısı 387.420.489 adet iç döngü kurmuş, bunlardan yalnızca 362.880 adedini kullanabilmiştir (1/1067).

9 şehir çözümü için yazılan bu kodun 9 şehri bulabilmesi için 6 saniye çözüm zamanı gerekmektedir.

Bu algoritma hesaplamayacağı dönümleri de gözden geçirdiği için etkili bir algoritma değildir.

2.1.2.5.6.3.1.2 Özyinelemeli Yapı

Özyinelemeli (recursive) fonksiyonlar belirli bir kritere ulaşıncaya kadar sürekli kendini çağırarak çalışırlar. Bu yapı kullanılarak daha verimli bir şekilde tüm olasılıkların bulunması sağlanmıştır.

Özyinelemeli yapıdaki algoritmanın temel iki fonksiyonu [27] :

```
void enumerate (char a[], int n)
{
        int n;
        if (n==0)
                printf ("%s\n", a);
        else
                for (i=0 ; i < n ; i++ )
                        {
```

```
                    swap (a, i, n-1);
                    enumerate (a, n-1);
                    swap (a, n-1 , i );
                    }
}
void swap (char a[] , int i, int j)
{
        char t;
        t=a[i];
        a[i]=a[j];
        a[j]=t;
}
```

Bu fonksiyonlar aşağıdaki C kodu ile 0,1,2,3 gibi dört şehir için çağrıldığında Şekil 2.6'daki gibi bir sonuç elde edilir.

```
int a[]= {0,1,2,3};
enumerate (a,4);
```

```
1.2.3.0
2.1.3.0
2.3.1.0
3.2.1.0
1.3.2.0
3.1.2.0
3.2.0.1
2.3.0.1
2.0.3.1
0.2.3.1
3.0.2.1
0.3.2.1
1.3.0.2
3.1.0.2
3.0.1.2
0.3.1.2
1.0.3.2
0.1.3.2
1.2.0.3
2.1.0.3
2.0.1.3
0.2.1.3
1.0.2.3
0.1.2.3
```

Şekil 2.6 Rekürsif algoritma ile 0, 1, 2, 3 rakamlarının diziliş alternatifleri

2.1.2.5.6.3.2 Problemin Çözülebilmesi İçin Gerekli Zaman Problemi

GSP probleminin sıralı çözüm algoritması, AMD 2.26 GHz işlemcili, UNIX işletim sistemi yüklü bilgisayarda çalıştırılarak sonuçlar elde edilmiş, geçen süreler Tablo 1'de verilmiştir.

Tablo 1 Problemin bir bilgisayar ile çözülebilmesi için gereken zamanlar

Şehir adedi	İhtimal adedi	Toplam hesaplama zamanı
<10	-	< 1sn
10	3.628.800	1 sn
11	39.916.800	12 sn
12	479.001.600	2da 14sn 96salise= 134.463sn
13	6.227.020.800	30da 47sn 83 salise = 1841.005 sn
14	87.178.291.200	7sa 34 da 75sn = 27.315sn
15	1.307.674.368.000	~121sa 48da 43 sn = 438.523sn
16	20.922.789.888.000	~ 2,7ay = ~80 gün

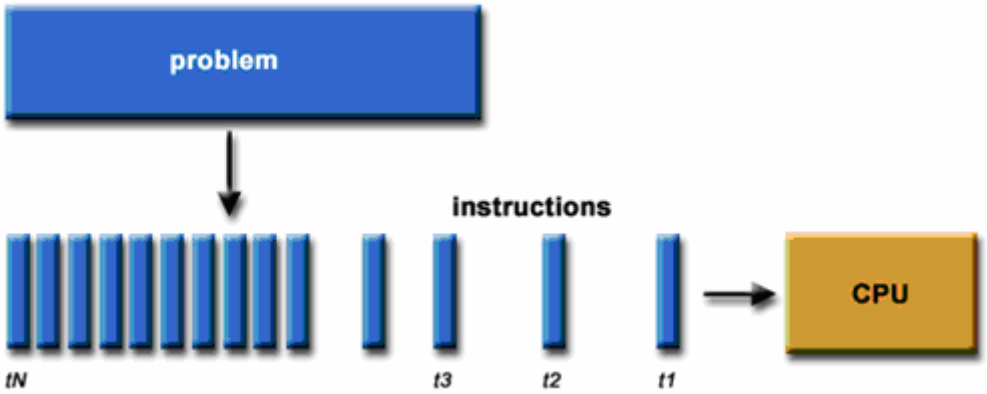
Tablo 1'den görüldüğü gibi, bahsedildiği üzere 16 adetten fazla şehir içeren bir GSP probleminin günümüz bilgisayarları ile çözümü, gerekli uzun hesaplama süresi nedeniyle imkansız hale gelmektedir.

Programın eski versiyonunda her bir ihtimal bulunduğunda, her bir ihtimal için rota uzunluğu hesabı yapılırken ikinci versiyonunda tüm şehir ikililerinin birbirine uzaklıkları iki boyutlu bir dizide saklanmış, her olasılıkta ilgili dizi indisinden hesaplanmış değerler okunarak çalışma süresi kısaltılmıştır. Bu şekilde algoritma optimize edilmiştir.

Örneğin birinci versiyonda 11 şehir 39.916.800 ihtimal için yaklaşık 20 saniye (19.672u 0.000s 0:30.49 64.5% 10+197k 0+1io 0pf+0w), ikinci versiyonda yaklaşık 10 saniye (10.565u 0.000s 0:10.60 99.6% 10+192k 0+1io 0pf+0w) almaktadır.

3. SERİ HESAPLAMA

Bilgisayar programları genel olarak sıralı olarak çalışacak şekilde seri algoritma ile yazılırlar. Bu algoritmalarda sıralı komutlar birbirini takip eder. Sırası gelen komut işlenir, yerini bir sonraki komuta bırakır. Sıralı komutlar çalıştığında program görevini icra etmiş olur.



Şekil 3.1 Seri Hesaplama

Teknoloji geliştikçe insan hayatında karşılaşılan bu gibi problemlerin karmaşıklığı artmaktadır. Bilim adamları karşılaştıkları matematiksel problemlere analitik çözümler aramakta, bu çözümler karmaşıklığa girdiğinde analitik çözümleri yetersiz kalmaktadır. Bu durumda bilgisayarlara başvurulur.

Bilgisayarların işlem gücü fazladır fakat günümüz hayat problemlerinde yetersiz kalabilmekte, işlemin çözümü için saatler, aylar, yıllar süren hesaplama zamanına ihtiyaç duyulmaktadır.

Hesaplama zamanındaki bu zaman ve güç gereksinimin önüne geçebilmek için döngü ve dizilerde optimizasyona gidilebilinse de bize istediğimiz zamanı

kazandıramayabilir. Büyük problemi daha kısa sürede çözebilmek için çoklu bilgisayar gücü (paralel işlem teknolojisi) kullanılmaktadır.

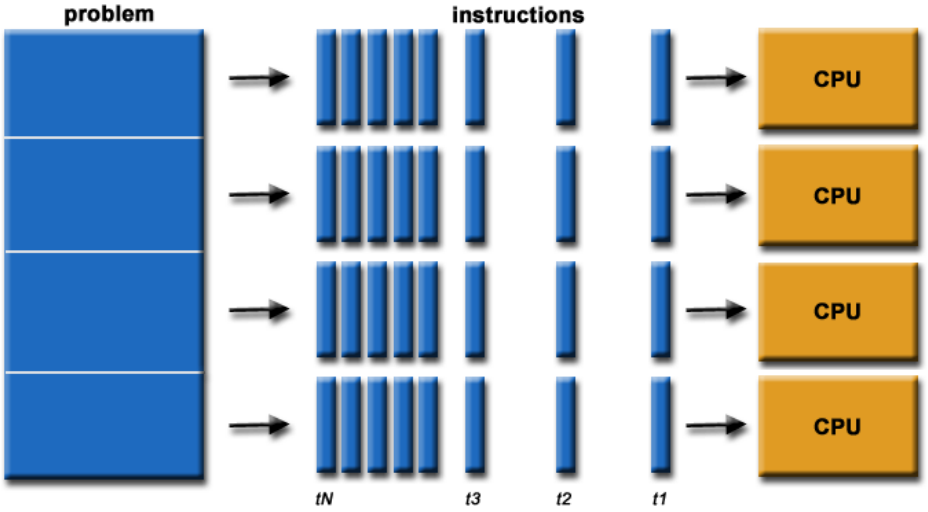
Paralel hesaplamanın birçok yöntemi vardır. Bunlardan birisi birden fazla işlemciye sahip ve zor problemleri çözmek için tasarlanmış olan süper bilgisayarları kullanmaktır. İlk olarak problemi inceleriz. Eğer problemimiz parçalanmaya elverişli ise her parçayı farklı işlemcilerde çalıştırabilirsiniz. Fakat sözü edilen süper bilgisayarların maliyeti günümüzde oldukça fazladır.

İkinci yöntemde de bahsedilen işlem gücünü ucuza mal edebilmek için ağ ortamı kullanma yoluna gidilmiştir. Birçok tek işlemcili bilgisayar ağ ortamında birleştirilerek güçlü bir yapı oluşturulmuştur. Parçalanan işin her bölümü birbirleri ile çok defa veri alışverişi yapacaksa burada ağın hızını da göz önüne almamız gerekmektedir [28].

4. PARALEL HESAPLAMA

Bir problemin çözülebilmesi için çeşitli bilgisayar kaynaklarının eşzamanlı olarak kullanılmasıdır. Bu yöntem ile işlenecek veri, bölünerek paralel olarak hesaplandığı için hız kazanılmıştır.

Paralel hesaplama, bir bilgisayar programının daha kısa bir sürede çalıştırabilmek için, program komutlarının veya program parçalarının birden fazla işlemci arasında bölüştürülmesi işlemidir [29].



Şekil 3.2 Paralel Hesaplama

Eş zamanlı çalıştırılabilecek işlerin, ayrı işleyiciler üzerinde (aynı bilgisayar veya farklı bilgisayarlarda) çalıştırılmasıdır [30].

Paralel hesaplama, çözülmesi uzun zaman gerektiren problemlerin daha kısa sürede çözülebilmesi için bulunmuş bir yöntemdir.

Hesaplaması uzun zaman alan, yüksek işlemci gücü gerektiren bazı problemler:

- Görüntü işleme,
- Veritabanı işlemleri,
- Sismik bilim,
- İklim bilimi,
- Sıvı akışkanlığı,
- Yüksek enerji fiziği,
- Genetik araştırmalar,
- Moleküler araştırmalar,
- Yarı iletken ve süper iletkenler,
- Uzay araştırmaları,
- Savunma sanayi,
- World Wide Web.

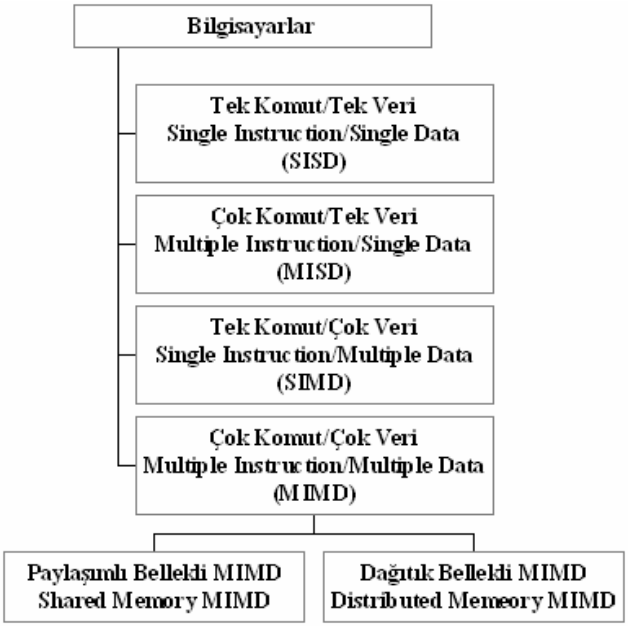
Paralel hesaplama mantığında, ağır hesaplama işlemini aynı anda bir işlemciye yaptırmaktansa, programı parçalara bölüp bu her parçayı değişik işlemcilere veya bilgisayarlara yaptırmak vardır [31].

Paralel hesaplama, programın çalışma hızının, onun eş zamanlı olarak yerel işlemcilerde çalışan parçalara bölünmesi ile artması fikriyle ilgilenir. *n* işlemcide çalıştırılan program tek işlemcide kullanıldığından *n* kez daha hızlı çalışır [32].

Paralel hesaplamayı cazip kılan nedenler [33] :

- Büyük problemlerin seri algoritmalarla çözümünün uzun zaman alması,
- Maliyetten kazanç, süper bilgisayarlara nazaran maliyeti ucuz hesaplama kaynaklarının varlığı,
- Tek bilgisayarın kısıtlı hafızasının sağladığı dezavantajın üstesinden gelmek için küme bilgisayar hafızalarının kullanılması,
- Eş zamanlılığın sağlanması, aynı zamanda çok uygulamanın yapılabilmesi.

4.1 Flynn Sınıflaması



Şekil 4.1 Flynn Sınıflaması

SISD (Single Instruction Single Data) Model :

Tek işlemci tek bir yönerge çalıştırır ve veri üzerindeki işlem tek bir hafıza üzerine yüklenir. İşlemcide bir anda bir adet işlem çalıştırılabilir.

SIMD (Single Instruction Multiple Data) Model :

Her bir işlem aynı komutu farklı veriler ile çalıştırmaktadır.

MISD (Multiple Instruction Single Data) Model :

Birden fazla işlem birimi aynı veri üzerinde farklı komutlar gerçekleştirir. Boru hattı mimarisi (pipeline architecture) bu tiptendir. Her bir elemanın çıktısı bir sonraki elemanın girdisi olacak şekilde sıralanmış işlemler zincirine "boru hattı (pipeline)" denir. Genellikle, ardışık birimlerin aralarına bir miktar arabellek (buffer) konulmaktadır. Bu boru hatları arasındaki bilgi akışı çoğunlukla bayt katarları (stream) ve bit katarları şeklindedir.

MIMD (Multiple Instruction Multiple Data) Model :

Her bir işlem farklı komutu farklı veriler ile çalıştırmaktadır. İleri düzey hesaplama sistemleri bu yapıyı kullanmaktadır [34].

Paralel işleme, paralel hesaplama hesaplanacak problemi çözmek için çoklu hesaplama kaynaklarının eş zamanlı kullanımıdır.

- Çoklu işlemci kullanılarak çalıştırılır,
- Kesikli parçalara parçalanmış bir problem aynı zamanda çözülebilir,
- Her parça seri yönergelere parçalanır,

Her parçanın yönergeleri farklı işlemcilerde eş zamanlı olarak çalışır.

Hesaplanabilir problem şu şekillerde olabilir :

- İşin eş zamanlı çözülebilecek kesikli parçalara ayrıştırılmış,
- Zamanın belirli bir anında çalışabilir çoklu program yönergeleri.

Paralel hesaplama kaynakları şunlardan oluşabilir :

- Süper bilgisayar denilen özel tasarlanmış, çoklu işlemciye sahip tek bilgisayar (Super Computer),
- Bir ağ ortamındaki bilgisayarların çok işlemcili bir bilgisayar gibi çalışacak hale getirilen bilgisayar kümesi (Beowulf Cluster),
- Her ikisinin karışımı [33].

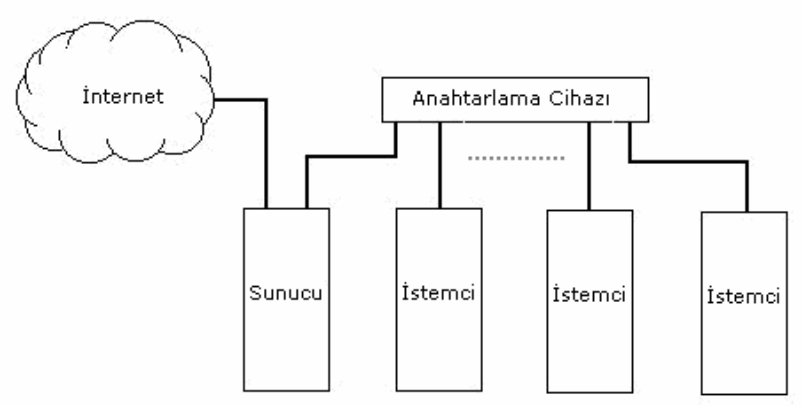
4.2 Beowulf

Beowulf, düşük maliyetli çok sayıda bilgisayarın kullanıcılardan soyutlanarak bir araya getirilmesi ile oluşturulan yüksek başarımlı bilgisayar sistemidir [35].

1994 yılında Thomas Sterling ve Don Becker adlı iki araştırmacı, yüksek işlem hızı elde edebilmek için NASA'nın bir projesi için 16 işlemciyi Ethernet ağı üzerinden birbirine bağlayıp paralel hesaplama gerçekleştirebilen bilgisayar kümesi elde ettiler. Bu kümeye de Beowulf adını verdiler.

Beowulf tipi bir makine topluluğu, bilindik süper bilgisayarlara nazaran daha yüksek performans ortaya koyabilmektedir. Beowulf'un kaliteli mimarisi ve kolay kurulumuyla beraber sadece donanıma ücret ödenmektedir. Beowulf üzerinde çalışan programların büyük çoğunluğu da ücretsizdir.

Sunucu (master node) bütün grubu kontrol eder ve istemci düğümlerine (slave node) dosya paylaşımını gerçekleştirir. Ayrıca sunucu, istemcilerin konsoludur ve dış dünya ile iletişimlerini sağlar. İstemci bilgisayarlarlar sunucu bilgisayar tarafından yapılandırılır ve kontrol edilir. İstemci düğümleri sadece kendilerine sunucu tarafından söyleneni yaparlar [36].



Şekil 4.2 Beowulf Topolojisi

4.2.1 Avantajları

- Yüksek işlem gücü,
- Esnek sistem mimarisi,
- Beowulf topluluğu, diğer süper bilgisayarlara göre daha yüksek performans ortaya koyabilmesi,
- Yalnızca donanım maliyeti olması,
- Beowulf üzerinde çalışan programların büyük çoğunluğunun ücretsiz olması.

4.2.2 Dezavantajları

- Uygulamaların paralel işlemeye uyumlu olma zorunluluğu, paralel programlama algoritmanın ona göre yazılması,
- Ağ ortamındaki bazı güvenlik sorunları, yetkisiz erişim,
- Uygunsuz veri dağıtımı,
- Geniş dağıtık bilgisayar kümeleri, yabancı sitelerden giriş.

4.2.3 Beowulf Sisteminin Tercih Edilmesinin Nedenleri

- Ücretsiz işletim sistemi ve yazılımların varlığı,
- PC donanımının ucuz olması, elde bulunan donanımı kullanarak sistemin oluşturulabilmesi,
- Herhangi bir yerden alınabilen PC donanımlarının donanımsal yapı için yeterli olması,
- Heterojen PC kümesi ile paralelliğin elde edilebilmesi,
- Ağ topolojilerindeki mevcut hızın artışı ve maliyet düşüklüğü.

4.2.4 Beowulf'ta Performans

Performans;

- Sistem işlemci gücüne,
- Ağ trafiğine,
- Bir uç bilgisayara gönderilen hesaplanacak veri boyutu büyüklüğüne,
- Birim zamandaki aktif node sayısına,
- Problemin ve yazılan programın çoklu bilgisayarlara dağıtılabilirlik özelliğine bağlıdır.

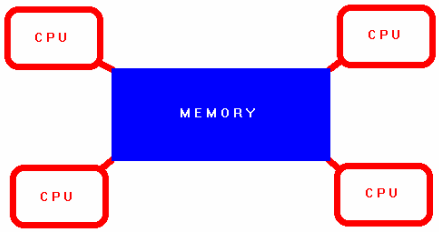
Testler sonucunda büyük boyutlu paketlerin kullanılmasının daha verimli olduğu görülmüştür [37].

4.2.5 Beowulf Topolojisinde Kullanılabilecek Hafıza Tipleri

İki çeşit hafıza kullanılabilmektedir. Paylaştırılmış hafıza mimarisinde genel ve ortak bir hafıza kullanılırken, dağıtılmış hafıza yapısında her bir hesaplama düğümü için ayrı bellek alanları tahsis edilmiştir [38].

4.2.5.1 Paylaştırılmış Hafıza (Shared Memory)

- Birden fazla işlemci aynı hafızayı kullanmaktadır,
- Bir anda yalnızca bir adet işlemci hafızaya erişebilir,
- Görevler arasındaki veri paylaşımı hızlıdır,
- Hafıza veri yolu sınırlıdır. İşlemci gücünü artırsanız bile hafıza veri yolunu artırmazsanız tıkanmalar yaşanabilir.



Şekil 4.3 Paylaşımlı hafıza

4.2.5.2 Dağıtılmış Hafıza (Distributed Memory)

- Tüm işlemciler kendi hafızalarında işlem yapmaktadır.
- Veri paylaşımı ağ bağlantıları ile mesajlaşma ile yapılmaktadır.
- İşlemci kadar hafıza vardır.
- Her işlemci büyük bir hızla arada bir ağ olmaksızın kendi belleklerine ulaşabilmektedirler.
- Bu dağıtılmış hafıza ortamında veri yapısının haritalanması oldukça zordur.

- Kullanıcı işlemciler arası gönderilen ve alınan veriden sorumludur. MPI gibi programlar ile gönderilen alınan veriler kontrol edilmelidir.
- Gecikmelerin önlenmesi için, karşı bilgisayarlar ihtiyaç duymadan veriler büyük paketler halinde hafizalara yüklenmelidir.

Şekil 4.4 Dağıtılmış hafiza

Basit cluster (küme) yapılarıyla başlayan Beowulf sistemleri git gide dünyaya yayılmış, global bir "GRID" halini almıştır.

4.3 GRID

Farklı coğrafi konumlardaki organizasyonlar aracılığı ile bilgisayarların hesaplama, uygulama, depolama ve ağ kaynaklarının paylaşılmasının bir yolu olarak tanımlanabilir. Web'in internet üzerinden bilgi paylaşımını sağlaması gibi, Grid de bilgisayar kaynaklarının ve depolama kapasitelerinin internet üzerinden paylaşımını sağlayan bir servis olarak düşünülebilir.

4.3.1 Grid Çeşitleri:

- CLUSTER GRID : Planlanması ve yerleştirilmesi kolay olan grid türüdür. Doğrudan erişime kapalı ortamlarda gerçekleştirildiği için güvenlik protokollerine fazla ihtiyaç duyulmamaktadır. Bu sebeple ortam kaynaklarından maksimum faydalanılabilmektedir.
- ENTERPRISE GRID : Bölüm veya Kampüs genişliğinde gruplara imkan tanımaktadır. Alt ağ seviyesinde bir güvenlik yeterli olmaktadır.

- GLOBAL GRID : Tüm dünya coğrafyasına dağılabilmektedir. Kaynaklar internet üzerinden paylaştırılır. İleri güvenlik politikaları ve protokollere ihtiyaç duyar.

4.4 Paralel Hesaplamada Kullanılan Yazılımlar

Kurulan bu sistemlerde bir programın çalıştırılabilmesi için bu programın alt yordamlara ayrılıp dağıtılması, bunların diğer bilgisayarlarda çalıştırılıp sonuçların tekrar birleştirilmesi gerekmektedir. Bu işlemler PVM, MPI, MOSIX gibi ek yazılımlar yardımıyla yapılır.

4.4.1 Parallel Virtual Machine (PVM)

Özel kütüphanesi olan bir yazılımdır, kodun paralel çalıştırılacak kısımları programcı tarafından belirlenir. Birden fazla işlemciye sahip tek bir bilgisayarda veya gerek yazılım gerekse donanımsal açıdan farklı özellikteki bilgisayar topluluğunun paralel çalışmasını sağlar.

4.4.2 Message Passing Interface (MPI)

Birden fazla bilgisayarın farklı işlemciler, farklı hafızalar üzerinde paralel programlama için bir kütüphanedir. Paralel programlama yükü kullanıcıya bakmaktadır. Kullanıcı paralelleştirmede kullanılan fonksiyonları kodunun içinde kullanarak verileri diğer bilgisayarlara göndererek işletir ve sonuçları toplar.

4.4.3 Mosix

Ücretli bir paralel processing yazılımıdır. Gönderilen bir iş otomatik olarak paralelleştirirlir.

4.5 Paralel Hesaplama Nelerle Uğraşır

Seri hesaplamanın değişmiş hali olan paralel hesaplama genellikle yüksek hesaplamayı göze alır. Grand Challenge Problems denilen çözülmesi zor ve hesaplama gücü gerektiren aşağıdaki hayat problemlerini çözmeye çalışır [33] :

- Gezegenlerle ve galaktik yörüngeler,
- Hava ve okyanus örnekleri,
- İklim bilimi,
- Kimyasal ve nükleer reaksiyonlar,
- Biyolojik, insan genleri,
- Jeolojik, sismik hareketler,
- Mekanik sürücüler , uzay araçları,
- Elektronik devreler,

Aşağıdaki örnek uygulamalar da daha fazla yüksek işlem gücü gerektirmektedir:

- Paralel veritabanları, veri madenciliği,
- Petrol araştırmaları,
- Arama motorları, Web tabanlı iş servisleri,
- Tıpta bilgisayar destekli tanı,
- Ulusal veya çok uluslu şirketlerin yönetimi,
- Özellikle eğlence sanayisinde ileri düzey grafik, 3 boyutlu görsellik,
- Multimedya teknolojileri,
- İşbirliği ile oluşturulan iş çevresi,
- Montaj hattı.

4.6 Paralel Hesaplamanın Farklı Yönü

Paralel hesaplamada, işlemi yapan işlem elemanı sayısı bir tane olmadığı için hesaplanacak iş sıralı (sequential) olarak çözülmemektedir. Hesaplanacak işlem

belirli parçalara bölünüp, her parça da ilgili hesaplama düğümüne gönderilerek yapılır.

Bu nedenle bir işlemci işin bir kısmını yaparken aynı anda diğer işlemci başka bir kısmını yapar. Böylece işlem adımları sırasıyla değil, parçalar halinde eş zamanlı olarak hesaplanıp zamandan kazanım sağlanmış olmaktadır [33].

Sadece uygun donanımı satın alıp, bağlantılarını yaptıktan sonra paralel hesaplamanın başarı şekilde gerçekleştirileceği asla düşünülmemelidir. Fiziksel alt yapının yanında yazılımsal olarak ta paralelizasyon yapılmalıdır.

Pratikte işlemci sayısı ile orantılı olan lineer hızlanmayı başarmak oldukça zordur. Ekstra işlemciler eklendikçe, bazı iş yükleri, boru hattı (pipeline) paralellik kullanarak belli bir noktaya kadar fayda sağlar. Bu sistem, bir fabrika montaj hattı yaklaşımı kullanarak işleri parçalar. Eğer iş *n* aşamaya bölünebiliyorsa ve bir ayrık değişken bir aşamadan diğer birine iletilebiliyorsa, en fazla *n* adet işlemci kullanılabilir. Bununla birlikte, en yavaş aşama diğer aşamaları da tutacaktır ve *n* işlemciyi tam performansta kullanmak pek mümkün olmayacaktır.

Pek çok algoritma, paralel donanımın kullanımını daha verimli yapmak için tekrardan tasarlanmalıdır. Tek işlemcili sistemlerde iyi çalışan programlar, paralel sistemlerde aynı performansı vermeyebilir. Aynı programın çoklu kopyaları, aynı anda aynı hafıza adresine yazma/okuma yapma nedeniyle birbirlerini etkileyebilirler. Bu yüzden paralel sistemlerde dikkatli programlama yapılması gerekir [39].

4.7 Amdahl Teorisi

Program hızı, parçalanmasıyla ilişkilendirilmiştir [38].

$$hızlanma = \frac{1}{1-p} \tag{4.1}$$

Eğer program paralelize edilmediyse $p=0$ olacak ve hız aynı kalacaktır.

Eğer programın tümü paralelize edildi ise *p=1* olur ve teorik olarak hız çok iyi değere sahip olur. Matematiksel olarak hızlanma değerinin sonsuz olduğunu görmekteyiz.

Eğer programın %50'si paralelleştirildi ise *p=0.5* olacağından, hızlanma değeri *2* olur. Yani iki kat hıza çıkabiliriz.

n işlemci ile elde edilen hızlanmayı gösteren $\Re(n)$, (4.2) formülü ile hesaplanabilir [41]:

$$\Re(n) = \frac{T(1)}{T(n)} \quad , \qquad T(a): \ a \text{ adet işlemci ile hesaplama süresi} \tag{4.2}$$

Elde edilecek verim, (4.3) ile hesaplanabilir [41].

$$\partial = \frac{T(1)}{n \cdot T(n)} \tag{4.3}$$

Paralel parça (*p*), işlemci sayısı (*n*) ve seri parçaya (*s*) bağlı olarak hızlanma, (4.4) formülü ile hesaplanabilir.

$$\Re(n) = \frac{1}{\frac{p}{n} + s} \tag{4.4}$$

n	P =0.50	P = 0.90	P = 0.99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02

Şekil 4.5 Örnek Hızlanma Değerleri

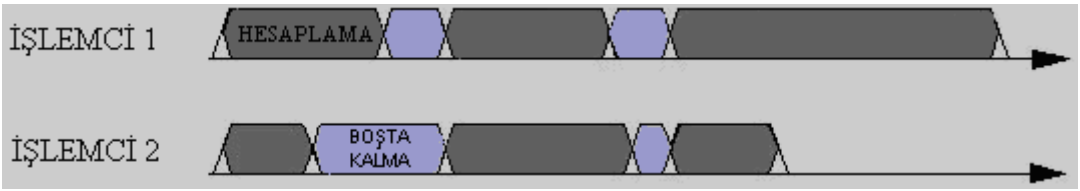
Şekil 4.5'ten paralelizmin ölçülebilirliğinin limitli olduğu açık bir şekilde görülebilir.

4.8 Verimsizlik

Bir uygulama bir birim zamanda bitiyor ise *n* adet bilgisayar kullanarak uygulamamızı çalıştırırsak teorik olarak *6/n* gibi bir zamanda bitmesi gerekir.

Paralel çalıştırdığımız iş en fazla *1/n* kadar zamanda bitmesi gerekirken, kullanılan yanlış algoritmalar, haberleşme fonksiyonlarının doğru kullanılmaması durumunda çıkmaza girme, yanlış veri alışverişi, gereksiz yere çok fazla veri transferi gibi nedenlerden dolayı *1/n* 'den daha fazla bir sürede bitebilmektedir.

Diğer bir önemli sebep ise bilgisayarlar arası iletişimin fazla olmasıdır. Algoritma geliştirilirken bilgisayarlar arasında gereksiz haberleşmeleri en aza indirgemek gerekmektedir. Bunun yanında haberleşmeler azaltılırken bilgisayarlar en fazla işlemi yapacak şekilde algoritma geliştirilmelidir. Gereksiz yere veri transferi, gereksiz yere haberleşme, bazı proseslerin az iş yapması, bir proses cevap beklerken diğer prosesin o işleme daha yeni başlaması vb. gibi problemler, paralel işlemlerden beklenen performansa engel olmaktadır [40]. Şekilde bir işlemci çalışırken diğer işlemcinin boşta kaldığı anlar görülmektedir.



Şekil 4.6 İşlemcinin hesaplama ve boşta kalma durumu

4.8.1 Kod Optimizasyonu İçin Düşünülen Algoritmalar

Şekil 4.6'da görülen koordinasyon eksikliğini giderebilmek ve gereksiz bekleyişleri önlemek için bir takım algoritmalar geliştirilmiştir.

4.8.1.1 İşleri Sıraya Sokma Algoritması (Scheduling Algorithm)

Bu algoritmada işlemlerimiz değişik parçalara bölünmektedir ve proseslerin alacağı işlemler tanımlanarak gereksiz yere bekleme veya gereksiz yere fazla iş yapma gibi problemlerin önüne geçilmektedir.

4.8.1.2 İndirgeme (Decomposition)

Birden fazla birbirine bağımlı işlemlerde sonuçları tek parça yerine parça parça göndermeye dayanır.

4.8.1.3 Yük Dağılımı

- Algoritmayı geliştiren, programın akışını önceden kestirerek hangi bilgisayara ne kadarlık iş yükleyeceğini hesap eder (Statik Yük Dağılımı).
- Ana proses sonuçları toplamaktadır ve birim zamanda gelen veri sayısına göre proseslere daha az veya daha çok iş yüklemektedir. Efendi-köle ilişkisinde prosesler oluşturup efendilerin belirlediği kadar köle prosesler çalıştırılmaktadır (Dinamik Yük Dağılımı).

Böylece prosesler arasında eş zamanlama oluşmaktadır. Dinamik yük dağılımında görüleceği gibi dinamik yönetim statik yönetimden daha karmaşıktır.

DOBSON modelinde kutuplardaki fırtına simülsayonu paralel sistemle incelenmiş ve hız artım istatistiği yapılmıştır.

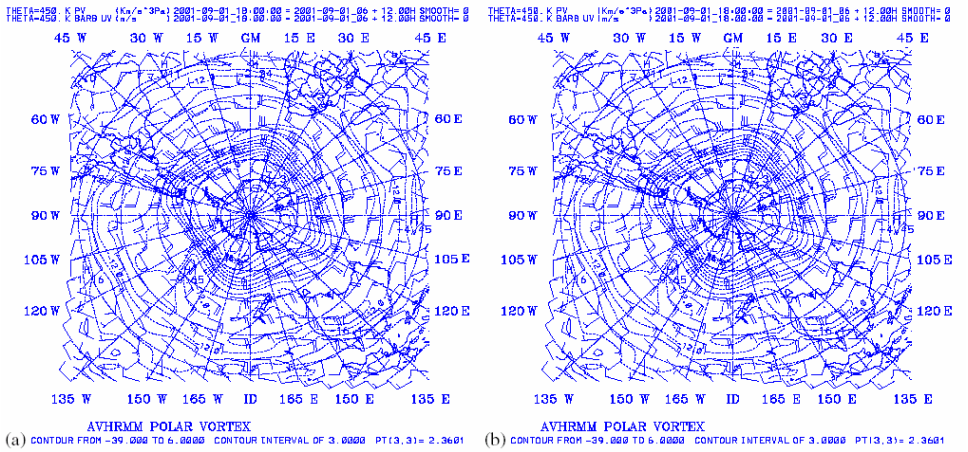
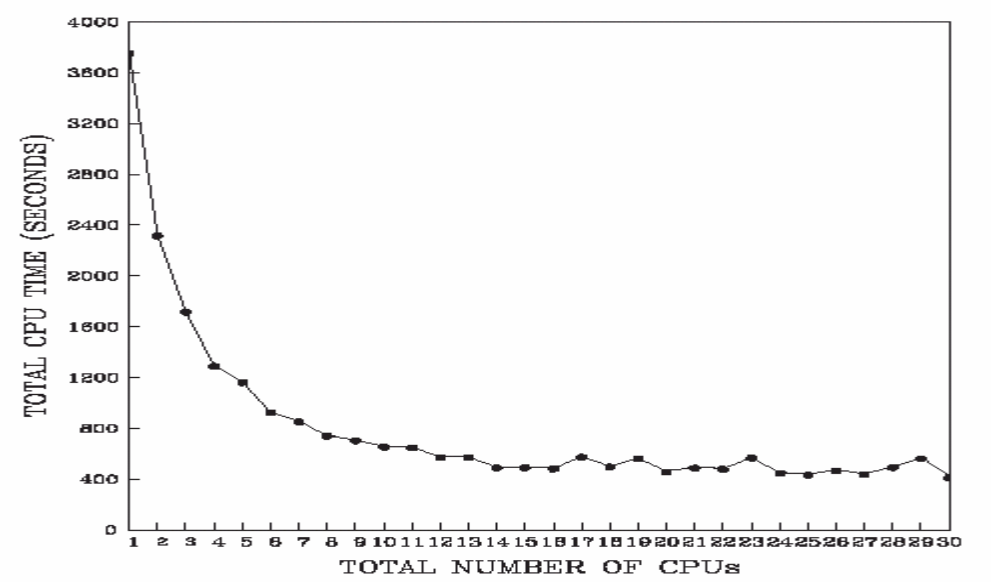


Fig. 1. Comparing 1-CPU (a) with 30-CPU (b) calculations of potential vorticity (dashed lines) and wind vectors at 450 K isentropic surface for a case simulation of the Antarctic polar vortex.

Şekil 4.7 DOBSON simülasyonu



Şekil 4.8 DOBSON modeli – İşlemci sayısına karşılık gerekli zaman karşılaştırması

Bu testte, hesaplama kümesine istediğinilen kadar hesaplama elemanı (işlemci) eklenemeyeceği, eklense bile sürekli artan bir performans yakalama ihtimalinin kesin olmadığı görülmektedir.

1 ile 8 işlemci arasında (5.1 kat) hız artımı var iken 1 ile 16 işlemcide arasında ancak (8.4 kat) hız sağlanabilmiştir. 17 işlemci kullanıldığında ise, 16 işlemciye göre daha uzun çalışma zamanı gereksinimi görülmüştür.

Tablo 2 Dobson modeli sonuçlarının incelenmesi

İşlemci Sayısı	Toplam İşlem Zamanı	İşlem Zamanındaki Hızlanma	Çalışma Yüzdesi
1	3800	1.00	1.00
2	2300	1.65	0.83
4	1300	2.92	0.73
8	750	5.07	0.63
16	450	8.44	0.53

5. ÇÖZÜLEBİLMİŞ EN İYİ GSP PROBLEMİ VE SONUÇLARI

Bugüne kadar çözülen en büyük gezgin satıcı problemi 24,978 noktalıdır ve İsveç'te yerleşimi olan her nokta için 96 bilgisayarlı bir ağ üzerinde 3 yılda çözülmüştür. Bu çözüm, Intel Xeon 2.8 Ghz bir işlemcinin 92 yılına denk bir sürede yapılmıştır.

Şu anda çözülmeye çalışılan en büyük problem Dünya üzerinde kayıtlı yerleşim olan her nokta için en kısa yolun ne olduğudur. Bu problem 1.904.711 şehir içermektedir [42].

6. ÖRNEK PROBLEMİN SERİ VE PARALEL HESAPLAMA SONUÇLARI

Problemlerin çözümü için kullanılan program, C programlama dilinde yazılmıştır. Bu program "2.1.2.5.6.3.1.2 Özyinelemeli Yapı" başlığı altında belirtilen yapının üzerine kurulmuştur.

C programlama dilinde yazılmış bir programının sistemde çalışabilmesi için onun çalıştırılabilecek derlenmiş dosya halinde olması gerekir. Bunun için C programlama dilinde yazılmış programın, C derleyicisi ile derlenmesi gerekir.

mpicc ornek.c –o derlenmis_dosya

Burada *ornek.c* isimli program dosyası, *mpicc* derleyicisi kullanılıp derlenerek, *derlenmis_dosya* isminde komut satırından çalıştırılabilir dosyaya çevrilmiştir.

Komut satırından *./derlenmis_dosya* komutu ile program çalıştırılmaktadır. Toplam hesaplama süresini ölçebilmek için UNIX sistemin desteklediği *time* komutu *time ./derlenmis_dosya* şeklinde kullanılmıştır.

Örnek bir çalıştırma sonucunda *time* komutunun çıktısı *0.893u 0.014s 0:00.91 98.9%* şeklinde olabilir.

6.1 Seri Hesaplama Sonuçları

Seri hesaplamalarda, AMD 2.26 GHz işlemcili, FreeBSD işletim sistemi ile çalışan bilgisayar kullanılmıştır.

5 şehir – 120 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]

SÜRE : 0.000u 0.001s 0:00.00 0.0% 0+0k 0+1io 0pf+0w

OPTIMUM ROTA >> **4** : [43,12] | **1** : [14,5] | **0** : [4,6] |**2** : [8,16] | **3** : [1,20]
OPTIMUM MESAFE >> **58.715332**

6 şehir – 720 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]

SÜRE : 0.000u 0.001s 0:00.01 0.0% 0+0k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **2** : [8,16] | **0** : [4,6] | **1** : [14,5] | **4** : [43,12]
OPTIMUM MESAFE >> **75.803337**

7 şehir – 5.040 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]

SÜRE : 0.002u 0.000s 0:00.00 0.0% 0+0k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **4** : [43,12]
OPTIMUM MESAFE >> **75.876564**

8 şehir – 40.320 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]

7.sehir [16,15]

SÜRE : 0.010u 0.000s 0:00.02 50.0% 8+144k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **7** : [16,15] | **4** : [43,12]
OPTIMUM MESAFE >> **83.407890**

9 şehir – 362.880 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]

SÜRE : 0.085u 0.000s 0:00.09 88.8% 11+198k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **7** : [16,15] | **8** : [43,18] | **4** : [43,12]

OPTIMUM MESAFE >> **89.407890**

10 şehir – 3.628.800 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]
9.sehir [22,26]

SÜRE : 0.893u 0.014s 0:00.91 98.9% 10+195k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **7** : [16,15] | **9** : [22,26] | **8** : [43,18] | **4** : [43,12]

OPTIMUM MESAFE >> **97.243904**

11 şehir – 39.916.800 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]

3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]
9.sehir [22,26]
10.sehir [6,18]

SÜRE : 10.815u 0.000s 0:10.84 99.7% 10+191k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **10** : [6,18] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **7** : [16,15] | **9** : [22,26] | **8** : [43,18] | **4** : [43,12] |

OPTIMUM MESAFE >> **97.395241**

12 şehir – 479.001.600 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]
9.sehir [22,26]
10.sehir [6,18]
11.sehir [40,8]

SÜRE : 134.463u 0.000s 2:14.96 99.6% 10+191k 0+1io 0pf+0w

OPTIMUM ROTA >> | **5** : [7,36] | **3** : [1,20] | **10** : [6,18] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **7** : [16,15] | **9** : [22,26] | **8** : [43,18] | **4** : [43,12] | **11** : [40,8]

OPTIMUM MESAFE >> **102.395241**

13 şehir – 6.227.020.800 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]
9.sehir [22,26]

10.sehir [6,18]
11.sehir [40,8]
12.sehir [30,50]

SÜRE : 1841.005u 0.000s 30:47.83 99.6% 10+191k 0+2io 0pf+0w

OPTIMUM ROTA >> | **12** : [30,50] | **5** : [7,36] | **3** : [1,20] | **10** : [6,18] | **2** : [8,16] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **7** : [16,15] | **9** : [22,26] | **8** : [43,18] | **4** : [43,12] | **11** : [40,8]

OPTIMUM MESAFE >> **129.321075**

14 şehir – 87.178.291.200 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]
9.sehir [22,26]
10.sehir [6,18]
11.sehir [40,8]
12.sehir [30,50]
13.sehir [21,5]

Süre: 7 sa 34 dk 75sn = 27315 sn.

Hesaplama Başlama Zamanı: Sun Jan 14 14:44:12 2007
Hesaplama Bitiş Zamanı : Sun Jan 14 22:19:27 2007

OPTIMUM ROTA >> | **12** : [30,50] | **5** : [7,36] | **9** : [22,26] | **7** : [16,15] | **2** : [8,16] | **10** : [6,18] | **3** : [1,20] | **0** : [4,6] | **6** : [10,6] | **1** : [14,5] | **13** : [21,5] | **11** : [40,8] | **4** : [43,12] | **8** : [43,18]

OPTIMUM MESAFE >> **135.435699**

15 şehir – 1.307.674.368.000 iterasyon
0.sehir [4,6]
1.sehir [14,5]
2.sehir [8,16]
3.sehir [1,20]
4.sehir [43,12]
5.sehir [7,36]
6.sehir [10,6]
7.sehir [16,15]
8.sehir [43,18]

9.sehir [22,26]
10.sehir [6,18]
11.sehir [40,8]
12.sehir [30,50]
13.sehir [21,5]
14.sehir [18,21]

Süre : 121sa 48da 43 sn = 438,523 sn.

Hesaplama Baslama Zamanı: Sun Jan 14 22:19:27 2007
Hesaplama Bitis Zamanı : Sat Jan 20 00:48:55 2007

OPTIMUM ROTA >> | **8** : [43,18] | **4** : [43,12] | **11** : [40,8] | **13** : [21,5] | **1** : [14,5] | **6** : [10,6] | **0** : [4,6] | **3** : [1,20] | **10** : [6,18] | **2** : [8,16] | **7** : [16,15] | **14** : [18,21] | **9** : [22,26] | **5** : [7,36] | **12** : [30,50]

OPTIMUM MESAFE >> **135.633408**

6.2 Paralel Hesaplama Sonuçları

6.2.1 Paralel Hesaplamada Kullanılan Problem Çözüm Mantığı

Paralel hesaplamayı gerçekleştirebilmek için mevcut problemin parçalanması gerekmektedir. GSP problemi, tam anlamıyla parçalanabilen bir problem değildir. Bir şehrin yerinin değişmesi, diğer tüm şehirleri de içine alan rotanın şeklinin değişmesine neden olmaktadır. Bu nedenle uygulamada kısmi bir parçalama yapılmıştır.

Eğer *n* adet şehir varsa, birinci işlem elemanına 1. şehir hariç tutularak diğer *n-1* adedi, ikinci işlem elemanına 2. şehir hariç tutularak diğer *n-1* adedi, .., *n.* işlem elemanına *n.* şehir hariç tutularak diğer *n-1* adedi için olası tüm ihtimaller her bir işlem elemanında hesaplanmaktadır. Bu şekilde aynı anda *n* adet ayrı hesaplanacak iş oluşmakta, bu işlerin her birisi ayrı hesaplama elemanında aynı anda (*n-1)!* şehir karmaşıklığında hesaplanmaktadır. Her bir hesaplama elemanında gerçekleşen işlem sonunda, o ünite için olası tüm alternatiflerde, her bir alternatif için bulunan rotanın uzunluğuna, rotanın son elemanı olan şehir ile hariç tutulan şehir arası uzaklık ta eklenerek toplam mesafe hesaplanır. Bu mesafelerin en küçüğü, o hesaplama

elemanının bulduğu en iyi rotanın uzunluğudur. Bu uzaklık değerini veren rotanın sonuna hariç tutulan şehir numarasının eklenmesi ile de o hesaplama elemanının bulduğu optimum rota elde edilmiş olunur.

En son işlem olarak her bir hesaplama elemanının bulduğu en kısa mesafeler arasından en kısa mesafe bulunarak optimum uzaklık ve optimum rota belirlenmiş olur.

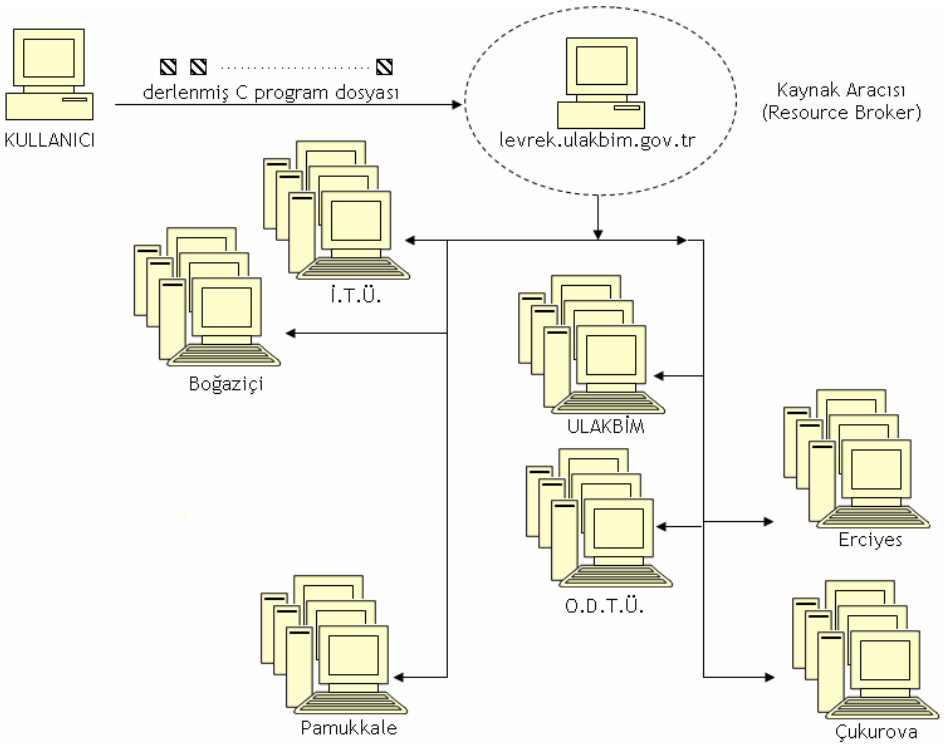
Bu şekilde problemin karmaşıklığı *n!* iken, (*n-1)!* 'e indirgenmiş olur. Toplam harcanan süre, *n-1* adet şehrin tüm alternatiflerinin tek işlem elemanında hesaplanması kadardır.

Hesaplama elemanlarına gönderilen derlenmiş program dosyaları tümünde aynı olmakla birlikte aldığı parametreler değişiklik göstermektedir. İlgili işlem elemanında hesaplama yapılırken hariç tutulacak şehir, gönderilen derlenmiş dosyaya parametre gösterilerek ifade edilmiştir.

./derlenmis_dosya 12 örneğinde, 12 numaralı şehir hariç tutularak diğer şehirler işleme tabi tutulur.

6.2.2 Paralel Hesaplama İçin Gerekli Donanımsal ve Yazılımsal Altyapı

Paralel hesaplamayı gerçekleştirebilmek için paralel bir sistem gerekmektedir. Bunun için grid sistemler kullanılabilmektedir. Grid altyapısı sunan bir kuruma başvurarak altyapıyı kullanmak için gerekli üyelik sağlanmalıdır. Burada ULAKBİM bünyesindeki Türkiye Grid'i (TR-Grid) kullanılmıştır.



Şekil 6.1 TR-Grid Hesaplama Merkezleri

Grid sistemin kimliğinizi algılayabilmesi için sertifika talebinde bulunmanız gerekmektedir. Talep sonunda grid sertifika otoritesi size bir sayısal sertifika vermektedir.

Grid yapıyı kullanabilmek için sayısal sertifika ve şifrenizi girerek bir proxy oluşturulmalıdır. Proxy, belirli bir zaman süresince sizin Grid'i kullanmanıza imkan sağlamaktadır. Oluşan proxy, Şekil 6.1'de görüldüğü gibi KULLANICI bilgisayarındaki hesaplanacak derlenmiş program dosyasının Kaynak Aracısına, Kaynak aracısından da sertifikanın geçerli olduğu üye hesaplama merkezlerine göndermesi için gereklidir.

Grid üzerinde değişik sanal organizasyonlar olabilir. Setifika otoritesinin izin verdiği sanal organizasyonlarda çalışılabilmektedir. Bunun için oluşturulacak Proxy'de sanal organizasyonun belirtilmesi gerekir.

Proxy oluşturabilmek için *glite-voms-proxy-init -voms X –valid Y* komutu verilmelidir. Bu komut içinde X yerine, kullanacağınız grid içindeki sanal

organizasyonun ismi, Y yerine ise proxy için saniye cinsinden geçerlilik süresi yazılmalıdır. Şekil 6.2'de, *trgridb* sanal organizasyonunda, geçerlilik süresi 240 saat olarak oluşturulmuş proxy görülmektedir.

```
mcetin@levrek:~$ glite-voms-proxy-init -voms trgridb -valid 240:00

Enter GRID pass phrase:
Your identity: /C=TR/O=TRGrid/OU=Balikesir/CN=Mustafa Cetin
Cannot find file or dir: /home_levrek/mcetin/.glite/vomses
Creating temporary proxy ....................................................... Done

Contacting  voms.ulakbim.gov.tr:15052 [/C=TR/O=TRGrid/OU=TUBITAK-ULAKBIM/CN=voms.ulakbim.gov.tr] "trgridb" Done

Warning: voms.ulakbim.gov.tr:15052: The validity of this VOMS AC in your proxy is shortened to 86400 seconds!

Creating proxy ................................................... Done
Your proxy is valid until Fri May 25 18:20:36 2007
```

Şekil 6.2 Proxy Oluşturmak

Proxy oluşturulduktan sonra, derlenmiş dosyanızı ilgili sanal organizasyona gönderebilirsiniz. İş gönderildiğinde, sanal organizasyondaki boş hesaplama ucundan birine yüklenmiş olur.

Derlenmiş bir dosyayı iş olarak gönderebilmek için JDL (Job Description Language) uzantılı işlem dosyaları kullanılmaktadır. Aşağıda *calistir.jdl* isminde örnek JDL dosyasının içeriği görülmektedir.

```
Executable = "derlenmis_dosya";
StdOutput = "cikti.out";
StdError = "cikti.err";
InputSandbox = {"derlenmis_dosya"};
OutputSandbox = {"cikti.out"," cikti.err"};
Arguments = "11";
```

Buradan da anlaşılabileceği gibi *derlenmis_dosya* ismindeki derlenmiş program dosyası, girdi birimi olarak kullanılmakta ve bu program dosyasının ekrana yazdırdığı tüm veriler *cikti.out* dosyasında saklanmaktadır. Oluşan hatalar ve time komutu çıktıları *cikti.err* dosyasına kaydedilmektedir. Arguments kısmında bulunan 11 değeri, *derlenmis_dosya* program dosyasının komut satırından aldığı parametreyi belirtmektedir.

JDL dosyası aracılığı ile derlenmiş C kodumuzun çalıştırılması için grid orta katmanlarından *glite* kullanılmıştır.

Grid üzerine iş göndermek için *glite-job-submit calistir.jdl* komutu verilmelidir. Her bir iş gönderildiğinde, o işe özel ve diğerlerine verilmemiş olan özel bir kimlik tahsis edilmektedir. Örneğin işi gönderdiğimizde, ona özel tahsis edilen kimlik kodu, *https://wms.ulakbim.gov.tr:9000/eNx7Lvznu048Vo1K-Fxk6w* gibi bir tekil (UNIQUE) kod olmaktadır.

İşin herhangi bir andaki durumunu öğrenebilmek için bu kimlik kodu kullanılmaktadır.

Verilen kimlik kodu görüldüğü gibi çok uzun karakterlerden oluşmaktadır. Bu nedenle işi başlatırken, -o parametresi ile işe bir isim verir, referans olarak o isim üzerinden işlem yaparız. ISLEMIM ismi ile iş gönderebilmek için kullandığımız komutu *glite-job-submit –o ISLEMIM calistir.jdl* şeklinde güncellenmelidir.

Gönderilen işin o andaki durumunu öğrenmek için *glite-job-status* komutu kullanılır. Örneğin yukarıdaki verilen iş kodu için şöyle yazılmalıdır:
glite-job-status https://wms.ulakbim.gov.tr:9000/eNx7Lvznu048Vo1KdFxk6w
Yukarıda bahsedildiği gibi, iş gönderilirken bir isim verilerek gönderilmiş ise komut *glite-job-status –i ISLEMIM* şeklinde yazılabilirdi.

Kullanıcının, tamamlanan işinin sonucunu karşıdan geri çağrılabilmesi için *glite-job-output –o ISLEMIM* komutu kullanılır.

Şekil 6.3'te bir işin *calistir5A.jdl* dosyası kullanılarak *15SEHIR_5nciUCv1* ismiyle karşıya gönderilmesi gösterilmektedir.

```
mcetin@levrek:~/JOBS/TAM_SONUCLAR/15$ glite-job-submit -o 15SEHIR_5nciUCv1 calistir5A.jdl

**** Warning: UI_VOMS_OVERRIDE ****
The Virtual Organisation name "dteam" you have specified  with the UI conf file will be overriden by the
default VO in your proxy credentials: "trgridb"

Selected Virtual Organisation name (from proxy certificate extension): trgridb
Connecting to host wms.ulakbim.gov.tr, port 7772
Logging to host wms.ulakbim.gov.tr, port 9002

================================ glite-job-submit Success ====================================
 The job has been successfully submitted to the Network Server.
 Use glite-job-status command to check job current status. Your job identifier is:

 - https://wms.ulakbim.gov.tr:9000/CHStrCrzGmtyeXZhzu7pYQ ..............> UNIQUE KİMLİK

 The job identifier has been saved in the following file:
 /home_levrek/mcetin/JOBS/TAM_SONUCLAR/15/15SEHIR_5nciUCv1
==============================================================================================
```

Şekil 6.3 Örnek bir iş gönderimi

Şekil 6.4'de işin kimlik kodu kullanılarak o anki durumunun öğrenilmesi gösterilmektedir.

```
mcetin@levrek:~/JOBS/TAM_SONUCLAR/16$ glite-job-status  https://wms.ulakbim.gov.tr:9000/CHStrCrzGmtyeXZhzu7pYQ

*************************************************************
BOOKKEEPING INFORMATION:

Status info for the Job : https://wms.ulakbim.gov.tr:9000/CHStrCrzGmtyeXZhzu7pYQ
Current Status:     Scheduled
Status Reason:      Job successfully submitted to Globus
Destination:        ce.ulakbim.gov.tr:2119/jobmanager-lcgpbs-trgridb
Submitted:          Wed Feb 14 23:04:50 2007 EET
*************************************************************
```

Şekil 6.4 Bir işin durumunun öğrenilmesi

Şekil 6.5'te, Şekil 6.4'te olduğu gibi kimlik kodu kullanmak yerine, o iş için tahsis edilmiş *15SEHIR_5nciUCv1* ismini kullanarak işin o andaki durumunun öğrenilmesi gösterilmektedir.

```
mcetin@levrek:~/JOBS/TAM_SONUCLAR/15$ glite-job-status -i 15SEHIR_5nciUCv1

*************************************************************
BOOKKEEPING INFORMATION:

Status info for the Job : https://wms.ulakbim.gov.tr:9000/CHStrCrzGmtyeXZhzu7pYQ
Current Status:     Scheduled
Status Reason:      Job successfully submitted to Globus
Destination:        ce.ulakbim.gov.tr:2119/jobmanager-lcgpbs-trgridb
Submitted:          Wed Feb 14 23:04:50 2007 EET
*************************************************************
```

Şekil 6.5 Bir işin durumunun öğrenilmesi - 2

Uygulamada kullanılan şehirlerin koordinatları Tablo 3'te belirtilmiştir.

Tablo 3 Seri ve Paralel Uygulamalar için kullanılan şehir koordinatları

ŞEHİR NUMARASI	X KOORDİNATI	Y KOORDİNATI
0	4	6
1	14	5
2	8	16
3	1	20
4	43	12
5	7	36
6	10	6
7	16	15
8	43	18
9	22	26
10	6	18
11	40	8
12	30	50
13	21	5
14	18	21
15	19	27

Tablo 4'te 12 şehir, Tablo 5'te 15 şehir, Tablo 6'da ise 16 şehir için paralel hesaplama sonuçları görülmektedir. Tablodaki son ek sütunu, işlem hesaplama anında hariç tutulan, sonradan hesaplamaya dahil edilen şehri göstermektedir.

Tablolarda en kısa ve en uzun sürede biten hesaplama zamanı değerleri koyu metin ile, optimum sonuçları veren rotalar ise koyu zemin ile gösterilmiştir. Optimum sonuçlar her tabloda ikişer tanedir. Bu durum optimum rotanın geri dönüşü olan ters istikametin maliyetinin de aynı olmasından kaynaklanmaktadır.

Çalışma zamanı sütununda görülen "user" ile belirtilen süre, programın çalışma zamanıdır. Sistem kaynakları dahil gerekli tüm zaman ise "real" olarak belirtilmektedir.

Tablo 4 12 şehir için, 11 şehir karmaşıklığında çözüm tablosu

Son Ek	Çalışma Zamanı	Opt. Uz.	Opt. Rota
0	real 0m9.223s user 0m9.220s	105.184959	11=4=8=9=5=3=10=2=7=1=6=0
1	real 0m9.229s user 0m9.230s	109.986923	11=4=8=9=5=3=10=2=7=0=6=1
2	real 0m9.421s user 0m9.410s	114.522057	11=4=8=9=5=3=10=0=6=1=7=2
3	real 0m9.091s user 0m9.090s	113.652351	11=4=8=9=5=7=1=6=0=2=10=3
4	real 0m9.442s user 0m9.440s	106.835548	5=3=10=2=0=6=1=7=9=8=11=4
5	real 0m9.093s user **0m9.080s**	102.395248	11=4=8=9=7=1=6=0=2=10=3=5
6	real 0m14.865s user 0m14.860s	111.111725	11=4=8=9=5=3=10=2=7=1=0=6
7	real 0m9.608s user 0m9.580s	107.893036	11=4=8=9=5=3=10=2=0=6=1=7
8	real 0m9.089s user 0m9.090s	105.378876	5=3=10=2=0=6=1=7=9=11=4=8
9	real 0m9.267s user 0m9.260s	110.420830	8=4=11=7=1=6=0=2=10=3=5=9
10	real 0m9.378s user 0m9.370s	114.117622	11=4=8=9=5=3=0=6=1=7=2=10
11	real 0m15.264s user **0m15.230s**	102.395241	5=3=10=2=0=6=1=7=9=8=4=11

Tablo 4'te görüldüğü üzere 12 şehir, son şehir ele alınarak 11 şehir karmaşıklığında (39.916.800 ihtimal) çözüm gerçekleşmiştir. Hesaplama süresi ise en yavaş düğümün çalışma zamanında (15.230 sn.) çözülmüştür. Seri hesaplama süresi göz önüne alındığında (4.2)'ye göre hızlanma, 134.463sn / 15.23sn = 8.83 olarak bulunur. Yani 8.83 kat hız elde edilmiştir. En hızlı hesaplama 9,08 sn. de gerçekleşmiştir.

Tablo 5 - 15 şehir için, 14 şehir karmaşıklığında çözüm tablosu

Son Ek	Çalışma Zamanı	Opt. Uz.	Opt. Rota
0	real 364m27.931s user **364m0.870s**	142.848053	12=5=3=10=2=7=14=9= 8=4=11=13=1=6=0
1	real 625m32.087s user **624m12.610s**	152.249680	12=5=3=10=2=0=6= 7=14=9=8=4=11=13=1
2	real 436m55.348s user 436m32.640s	152.185150	12=5=3=10=0=6=1=13= 11=4=8=9=14=7=2
3	real 365m34.511s user 364m54.520s	150.508453	11=4=8=12=5=9=14=7= 13=1=6=0=2=10=3
4	real 365m54.649s user 365m23.100s	141.07373	12=5=9=14=7=2=10=3= 0=6=1=13=11=8=4
5	real 365m16.829s user 364m50.020s	142.903885	8=4=11=13=1=6=0=3=10=2= 7=14=9=12=5
6	real 365m19.003s user 364m50.520s	148.774826	12=5=3=10=2=7=14=9= 8=4=11=13=1=0=6
7	real 366m24.475s user 365m58.800s	145.556122	12=5=3=10=2=0=6=1= 13=11=4=8=9=14=7
8	real 365m19.740s user 364m53.680s	135.633423	12=5=9=14=7=2=10=3= 0=6=1=13=11=4=8
9	real 436m48.491s user 436m30.680s	149.568710	11=4=8=12=5=3=10=2= 0=6=1=13=7=14=9
10	real 475m55.439s user 475m30.400s	151.780701	12=5=3=0=6=1=13=11= 4=8=9=14=7=2=10
11	real 625m43.646s user 624m4.840s	137.501099	12=5=3=10=2=0=6=1= 13=7=14=9=8=4=11
12	real 366m50.599s user 366m24.230s	135.633408	8=4=11=13=1=6=0=3= 10=2=7=14=9=5=12
13	real 364m30.726s user 364m4.220s	148.754166	12=5=3=10=2=0=6=1= 7=14=9=8=4=11=13
14	real 388m7.757s user 387m40.840s	151.523499	12=5=3=10=2=7=0=6= 1=13=11=4=8=9=14

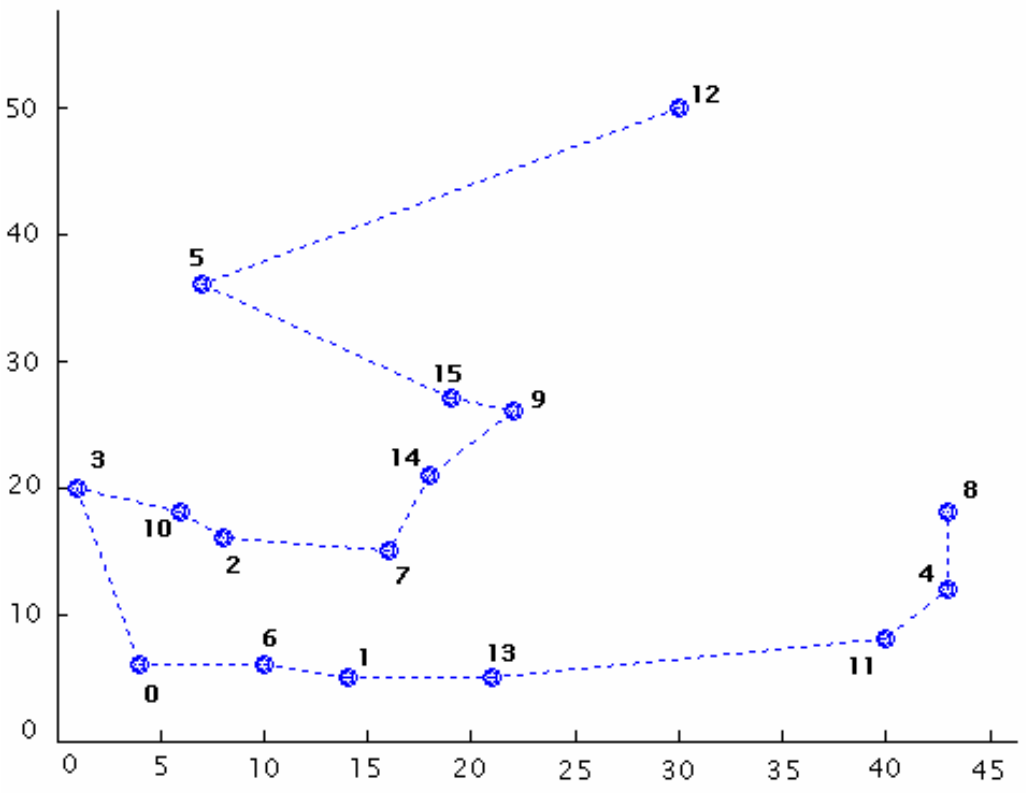
Tablo 5'e göre; 15 şehir, son şehir ele alınarak 14 şehir karmaşıklığında ve en yavaş düğümün hızında (624 dakika 12.61 sn =37452.61 sn) çözülmüştür. Seri hesaplama süresi göz önüne alındığında (4.2)'ye göre hızlanma, 438523sn / 37452.61sn=11.7 olarak bulunur. Yani 11.7 kat hız elde edilmiştir. En hızlı hesaplama 364 dakika 0.87 saniyede gerçekleşmiştir.

Tablo 6 - 16 şehir için, 15 şehir karmaşıklığında çözüm tablosu

Son Ek	Çalışma Zamanı	Opt. Uz.	Opt. Rota
0	real 6485m23.393s user 6479m51.450s	145.689972	12=5=3=10=2=7=14=15=9=8 =4=11=13=1=6=0=
1	real 6517m7.341s user 6511m54.080s	153.040543	12=8=4=11=13=7=14=9=15=5 =3=10=2=0=6=1=
2	real 6503m21.266s user 6497m35.470s	153.089874	12=8=4=11=13=1=6=0=7=14 =9=15=5=3=10=2=
3	real 6458m48.144s user 6456m0.670s	150.642975	11=4=8=12=5=15=9=14=7=13 =1=6=0=2=10=3=
4	real 6679m22.316s **user 6672m33.460s**	141.208252	12=5=15=9=14=7=2=10=3=0 =6=1=13=11=8=4=
5	real 6483m53.131s user 6479m25.490s	143.381943	12=8=4=11=13=1=6=0=3=10 =2=7=14=9=15=5=
6	real 6510m10.616s user 6505m44.960s	151.616745	12=5=3=10=2=7=14=15=9=8 =4=11=13=1=0=6=
7	real 6448m43.629s **user 6445m56.750s**	148.398041	12=5=3=10=2=0=6=1=13=11 =4=8=9=15=14=7=
8	real 6514m34.572s user 6509m54.000s	135.767944	12=5=15=9=14=7=2=10=3=0 =6=1=13=11=4=8=
9	real 6623m49.469s user 6617m13.580s	152.410629	11=4=8=12=5=3=10=2=0=6= 1=13=7=14=15=9=
10	real 6452m24.302s user 6448m49.760s	154.094040	12=8=4=11=13=1=6=0=2=7= 14=9=15=5=3=10=
11	real 6454m12.200s user 6451m2.910s	140.343002	12=5=3=10=2=0=6=1=13=7= 14=15=9=8=4=11=
12	real 6453m19.316s user 6450m33.760s	135.767929	8=4=11=13=1=6=0=3=10=2= 7=14=9=15=5=12=
13	real 6625m25.574s user 6620m36.060s	151.072388	11=4=8=12=5=15=9=14=7=2 =10=3=0=6=1=13=
14	real 6490m42.431s user 6487m37.310s	151.837021	12=5=15=9=8=4=11=13=1=6 =0=3=10=2=7=14=
15	real 6463m52.663s user 6459m44.610s	152.730988	11=4=8=12=5=3=10=2=0=6= 1=13=7=14=9=15=

Tablo 6'ya göre; 16 şehir, son şehir ele alınarak 15 şehir karmaşıklığında ve en yavaş düğümün hızında (4 gün 15 saat 12 dakika 33.46 saniye = 6672 dakika 33.460sn = 400353.46 sn) çözülmüştür. Yaklaşık 80 gün sürmesi beklenen hesaplama, 4.62 günde bitmiştir. En hızlı hesaplama 6445 dakika 56.75 sn.'de gerçekleşmiştir.

Şekil 6.6'de, Tablo 3'te koordinatları verilen 16 şehir için bulunan optimum rotanın grafiği görülmektedir. Turu 12 numaralı şehirden başlayıp, 8 numaralı şehirde bitirmek ile, 8 numaralı şehirden başlayıp 12 numaralı şehirde bitirmek eşit maliyetlidir.



Şekil 6.6 16 Şehirli Problem için Sonuç Grafiği

7. GENEL DEĞERLENDİRME VE SONUÇ

Gerçek hayattaki, GSM operatörleri baz istasyonu yerleşimi, ulaşım ve lojistik sorunları, üretim planlama, malzeme akış sistemleri, posta kutusuna dağıtım problemleri, araç rotalama problemleri, stok alanı yeri seçimi, malzeme toplama problemleri, uçaklar için havaalanı rota tayini, elektronik devre tasarımları gibi bir çok alanda kullanılan GSP probleminin en iyi çözümleri halen kıymetlidir.

Çok alanda kullanılan bu problemin optimal değerleri, o problemin çözümünde referans alınabilir.

Bu problemlerin kesin en iyi çözümlerinin bilinmesi, kısa sürede yaklaşık çözümleri araştıran sezgisel (heuristic) algoritmaların başarısını ölçmede performans kriteri olmaktadır.

Paralel hesaplamayı kullanmak, kesin sonuçlarının elde edilebilmesi için yüksek işlemci gücü ve zaman gerektiren problemlerin çözümünde etkili bir yöntemdir. Bu tip problemlerin çözümünde sıralı algoritmalar yetersiz kalmaktadır. Paralelleştirildikleri durumlarda makul sürelerde çözüm imkanı bulunabilmektedir.

Paralel hesaplama teknolojisi çözülemeyen problemleri eğer problemin yapısı müsait ise kabul edilebilir sürelerde çözmeye imkan tanımaktadır.

KAYNAKLAR:

[1] http://akademik.maltepe.edu.tr/~causlu/Yoneylem/Yoneylemegiris.ppt (erişim tarihi: 14.02.2007)

[2] Ramazan Yaman, Kadriye Ergün, Fırat Evirgen, Necati Özdemir, Gülşen Yaman Classification Approaches to Optimization Problems and a Discrete Optimization Case

[3] Optimizasyon Problemlerine NEOS yaklaşımı (NEOS, 2005)

[4] Ergün, K., Kesme ve paketleme problemleri ve araştırmaya yönelik bir metod geliştirilmesi ve bu metodun etkinliğinin sınanması, Yüksek Lisans Tezi, Balıkesir Üniversitesi Fen Bilimleri Enstitüsü, Endüstri Mühendisliği Anabilim Dalı, Balıkesir (2004)

[5] http://en.wikipedia.org/wiki/Combinatorial_optimization (erişim tarihi: 14.02.2007)

[6] http://w3.gazi.edu.tr/web/bdengiz/turkce/courses/enm543.html (erişim tarihi: 07.06.2005)

[7] http://iris.gmu.edu/~khoffman/papers/newcomb1.html (erişim tarihi: 09.06.2005)

[8] http://www.mmo.org.tr/endustrimuhendisligi/2004_2/bilkent.htm (erişim tarihi: 11.06.2005)

[9] http://www.tsp.gatech.edu/problem/index.html (erişim tarihi: 14.02.2007)

[10] Bauk, S., Kovac, N. , " The Comparative Analysis of two Neural Networks Models in The Function Of The Linear Ship's Route Costs Minimization ", *JEL clasification: L92*., (2006), 89

[11] http://www.yapay-zeka.org/modules/wiwimod/index.php?page=Travelling+Salesman&back=AI_Problems (erişim tarihi: 14.02.2007)

[12] Applegate,D., Bixby,R., Chvatal, V., Cook, W., "On the Solution of Traveling Salesman Problems", Documenta Mathematica, (1998), 645

[13] Uğur, A., A., Aydın, D., "Ant System Algoritmasının Java ile Görselleştirilmesi", ab.org.tr (2006)

[14] Wismer, D.A., Chattergy, R., "A Problem Solving Approach, Elsevier North Holland", ISBN 0-444-00234-0, Introduction to Nonlinear Optimization, (1979).

[15] http://tr.wikipedia.org/wiki/Dola%C5%9Fan_sat%C4%B1c%C4%B1 (erişim tarihi: 20.04.2007)

[16] Macit, İ., Endüstride Bilgisayar Uygulamaları Ders Notları, Çukurova üniversitesi (Adana),(2006).

[17] Bakır, M.A., Altunkaynak, B., Tamsayılı programlama, Nobel Yayın (Ankara), (298.Sayfa).

[18] http://www.baskent.edu.tr/~ikara/Karavd.ppt Genelleştirilmiş Gezgin Satıcı Probleminin Tamsayılı Karar Modeli (erişim tarihi: 14.02.2007)

[19] http://tr.wikipedia.org/wiki/Turing_Makinesi (erişim tarihi: 20.04.2007)

[20] http://tr.wikipedia.org/wiki/NP_%28karma%C5%9F%C4%B1kl%C4%B1k%29 (erişim tarihi: 20.04.2007)

[21] http://tr.wikipedia.org/wiki/P_ile_NP_aras%C4%B1ndaki_ili%C5%9Fki (erişim tarihi: 20.04.2007)

[22] Taşgetiren,F., Ders Notları, 9.Bölüm, www.fatih.edu.tr

[23] Sural, H., "Gezgin Satıcı Problemi", Matematik Dünyası dergisi, 2003 Güz, s:37-40

[24] Cedimoğlu, İ. H., Geyik, F., "Tabu Arama Tekniği ile Klasik İş-Atölyesi Çizelgeleme", YAEM'99-Yöneylem Araştırması ve Endüstri Mühendisliği XX. Ulusal Kongresi.

[25] http://www.mmo.org.tr/muhendismakina/arsiv/2001/ekim/Genetik_Algoritma.htm (erişim tarihi: 16.02.2007)

[26] Kalaycı, T.E., Yapay zeka teknikleri kullanan üç boyutlu grafik yazılımları için "extensıble 3d" (x3d) ile bir altyapı oluşturulması ve gerçekleştirimi, Yüksek Lisans Tezi, Ege Üniversitesi

[27] www.cs.princeton.edu/courses/archive/spr01/cs126/lectures/P7-4up.pdf (erişim tarihi: 14.02.2007)

[28] Kanmaz, A.A., "Paralel Hesaplama Kütüphaneleri (MPI, PVM) ve Linux Ağında Çalıştırılmaları", Akademik Bilişim 2003, 3-5.02.2003, Adana

[29] http://sozluk.sourtimes.org/ (erişim tarihi: 23.05.2005)

[30] http://ab2004.ktu.edu.tr/sunum/kyk1.pps (erişim tarihi: 23.05.2005)

[31] http://www.be.itu.edu.tr/kaynak/kaynak/sy/ (erişim tarihi: 23.05.2005)

[32] http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto-1.html#ss1.4 (erişim tarihi: 14.02.2007)

[33] http://www.llnl.gov/computing/tutorials/parallel_comp/ (erişim tarihi: 14.02.2007)

[34] http://en.wikipedia.org/ (erişim tarihi: 14.02.2007)

[35] http://ab2004.ktu.edu.tr/sunum/kyk1.pps (erişim tarihi: 23.05.2005)

[36] http://www.faqs.org/docs/Linux-HOWTO/Beowulf-HOWTO.html (erişim tarihi: 23.05.2005)

[37] http://seminer.linux.org.tr/konferanslar/inet-tr98/ppthtml/bs-beowulf/bs-beowulf.ppt (erişim tarihi: 23.05.2005)

[38] http://www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html (erişim tarihi: 23.05.2005)

[39] http://tr.wikipedia.org/wiki/Paralel_hesaplama (erişim tarihi: 23.05.2005)

[40] http://www.csharpnedir.com/makalegoster.asp?MId=503 (erişim tarihi: 23.05.2005)

[41] Karasulu, B., Aybars., U., Özörgütlemeli yapay sinir ağı modeli'nin kullanıldığı kutup dengeleme problemi için paralel hesaplama
tekniği ile bir başarım eniyileştirme yöntemi, Akademik Bilişim 2007, Kütahya

[42] http://tr.wikipedia.org/wiki/Seyyar_sat%C4%B1c%C4%B1_problemi (erişim tarihi: 20.04.2007)